ალგორითმების აგება

ლელა ალხაზიშვილი
ალექსანდრე გამყრელიძე

ნახაზები, მაგალითები, დავალებები და LaTeX: ლევან კასრაძე

სარჩევი

თავი 1

ძირითადი ცნებები გრაფთა თეორიიდან

1.1 შესავალი

კომპიუტერული მეცნიერების სპეციალისტისთვის ალგორითმების შესწავლა მნიშვნელოვანია როგორც პრაქტიკული, ისე თეორიული თვალსაზრისით. პრაქტიკულად მას უნდა ჰქონდეს წარმოდგენა ძირითად ალგორითმებზე, რომლებიც შესაბამის მონაცემთა სტრუქტურის არჩევით, პროგრამული ინსტრუმენტების გამოყენებით, მისცემს ალგორითმის ეფექტურად იმპლემენტაციის საშუალებას ამა თუ იმ ამოცანისთვის. თეორიული თვალსაზრისით, ალგორითმები წარმოადგენენ ბაზისს, რომლის გარეშეც შეუძლებელია პროგრამული კოდის დაწერა.
ალგორითმების შესწავლა არ გულისხმობს რამდენიმე ალგორითმის აღწერას რამდენიმე კონკრეტული ამოცანისთვის, არამედ სტილის და მიდგომების გააზრებას, რომელიც გამოადგება პროგრამისტს ახალი ამოცანის დასმისას, ალგორითმის გამოყენებაში. მეორე მხრივ, პროგრამისტმა უნდა შეძლოს გაარჩიოს ამოხსნადია თუ არა დასმული ამოცანა.
არსებობს ალგორითმების კლასიფიკაციის ორი მიდგომა: ამოცანის ტიპის და პროექტირების მეთოდის მიხედვით.
ამოცანის ტიპის მიხედვით ალგორითმები შეიძლება დაიყოს შემდეგნაირად:

- დახარისხება
- ძებნა
- სტრიქონის დამუშავება
- ამოცანები გრაფებზე
- კომბინატორული ამოცანები
- გეომეტრიული ამოცანები
- რიცხვითი ამოცანები

პროექტირების მეთოდის მიხედვით ალგორითმები შეიძლება დაიყოს:

- "უხეში ძალის" მეთოდი
- დეკომპოზიციის მეთოდი
- ამოცანის ზომის შემცირების მეთოდი
- გარდაქმნის მეთოდი
- დროისა და სივრცის კომპრომისის მეთოდი
- ხარბი მეთოდი
- დინამიკური დაპროგრამების მეთოდი
- დაბრუნებით ძებნის მეთოდი
- შტოებისა და საზღვრების მეთოდი

ჩვენს მიერ შესასწავლი ალგორითმების უმრავლესობას აქვს პოლინომიალური მუშაობის დრო, რაც ნიშნავს, რომ უარეს შემთხვევაში მუშაობის დრო არის $O(n^k)$. ასეთ ამოცანებს მიაკუთვნებენ P კლასს. უფრო ფორმალურად, P-ში შედის მხოლოდ **გადაწყვეტილების მიღების** ამოცანები (decision problems), ანუ ამოცანები, რომელზეც პასუხია "კი" ან "არა". მაგრამ ყველა ამოცანის ამოხსნა არ შეიძლება პოლინომიალურ დროში. არსებობს **გადაწყვეტილების მიღების** ამოცანები, რომელიც არ იხსნება საერთოდ, არც ერთი ალგორითმით. ასეთ ამოცანებს უწოდებენ **ამოუხსნადს** (undecidable). ამ ტიპის ამოცანის ცნობილი მაგალითია ალან ტიურინგის მიერ 1936 წელს დასმული გაჩერების ამოცანა (halting problem): მოცემული კომპიუტერული პროგრამისთვის და შემავალი მონაცემებისთვის, განსაზღვრეთ დაამთავრებს თუ არა პროგრამა მუშაობას, თუ ის იმუშავებს უსასრულოდ.
NP კლასში შედის გადაწყვეტილების მიღების ამოცანები, რომლებიც "ექვემდებარებიან შემოწმებას" პოლინომიალურ დროში. სხვა სიტყვებით რომ ვთქვათ, თუ გვაქვს ამ ამოცანის რაიმე ამონახსნი, პოლინომიალურ დროში შეიძლება შევამოწმოთ მისი კორექტულობა. P კლასის ნებისმიერი ამოცანა ეკუთვნის NP კლასს: $P \subseteq NP$. მაგრამ ჯერჯერობით (კუკის მიერ, მისი დასმის მომენტიდან, 1971 წ.) ღიად რჩება საკითხი: P კლასი NP კლასის მკაცრი ქვესიმრავლეა, თუ ეს ორი კლასი ემთხვევა ერთმანეთს?
ამოცანებისთვის, ამოცანათა კლასიდან, რომელიც ცნობილია NP-სრული კლასის სახელწოდებით, არ არის ნაპოვნი ამოხსნის ეფექტური ალგორითმები, მაგრამ ისიც არ არის დამტკიცებული, რომ ასეთი ალგორითმები არ არსებობს. მეორე მხრივ, NP-სრულ ამოცანებს აქვთ ერთი შესანიშნავი თვისება: თუ ეფექტური ალგორითმი არსებობს ერთი მაინც ამოცანისთვის ამ კლასიდან, მაშინ მისი ფორმულირება შეიძლება ამ კლასის ყველა დანარჩენი ამოცანისთვისაც.
NP-სრული ამოცანების განსაკუთრებული თვისება არის ის, რომ ზოგიერთი მათგანი, ერთი შეხედვით, ანალოგიურია ამოცანებისა, რომელთათვისაც არსებობს ალგორითმები პოლინომიალური მუშაობის დროით. მაგალითად, განვიხილოთ წყვილები, რომლებშიც ერთი იხსნება პოლინომიალურ დროში, ხოლო მეორე - NP-სრული ამოცანაა.

1. ეილერის და ჰამილტონის ციკლების:

   ეილერის ციკლის პოვნის ამოცანა: ვიპოვოთ G=(V,E) ბმულ ორიენტირებულ გრაფში ციკლი, რომელიც ყველა წიბოს შემოივლის მხოლოდ ერთხელ, თუმცა დასაშვებია წვეროების რამდენჯერმე შემოვლა. ეილერის ციკლის არსებობის დადგენა, აგრეთვე, მისი შემადგენელი წიბოების პოვნა შეიძლება O(E) დროში.

   ჰამილტონის ციკლის პოვნის ამოცანა: ვიპოვოთ G=(V,E) ორიენტირებულ გრაფში მარტივი ციკლი, რომელიც ყველა წვეროს შემოივლის. ჰამილტონის ციკლის არსებობის დადგენის ამოცანა არის NP-სრული.

2. გრაფში ორ წვეროს შორის უმოკლესი გზის და ყველაზე გრძელი გზის პოვნის ამოცანები:

   G=(V,E) ორიენტირებულ გრაფში ორ წვეროს შორის უმოკლესი გზის პოვნის ამოცანა შეიძლება ამოიხსნას O(VE) დროში. ორ წვეროს შორის ყველაზე გრძელი გზის პოვნის ამოცანა რთულია. ამოცანა იმის შესახებ, შეიცავს თუ არა გრაფი მარტივ გზას, რომელშიც წიბოების რაოდენობა მოცემულ რიცხვზე ნაკლები არაა, NP-სრულია.

   თუ რაიმე ამოცანისთვის დადგინდა, რომ იგი NP-სრულია, მაშინ პროგრამისტი უფრო ეფექტურად დახარჯავს დროს, თუ შექმნის ამ ამოცანისთვის მიახლოებით ალგორითმს ან ამოხსნის ამ ამოცანის მარტივ ვარიანტს, იმის მაგივრად, რომ ეძებოს სწრაფი ალგორითმი, რომელიც იძლევა ზუსტ ამონახსნს.

ალგორითმების აგების ამ კურსს ვიწყებთ გრაფებზე ალგორითმების შესწავლით, ამისთვის გავეცნოთ ძირითად ცნებებს და განსაზღვრებებს გრაფების შესახებ.

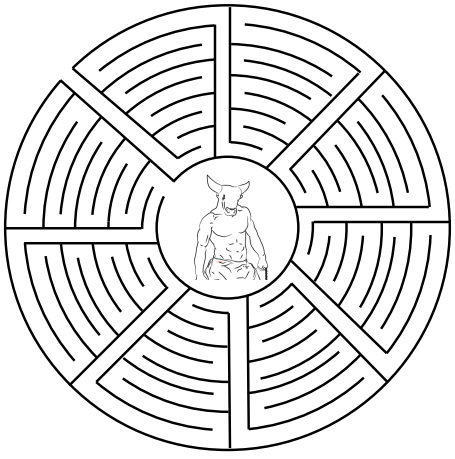
1.2 ძირითადი ცნებები და განსაზღვრებები გრაფებზე

როგორც წინა სემესტრის შესავალ კურსში აღვნიშნეთ, გრაფებით *ძალიან ბევრი* პრობლემის აღწერა და გადაჭრა შეიძლება. თუ მოცემულია რაიმე ამოცანა A, მისი მონაცემების გარდაქმნა შეიძლება ისეთ გრაფად, რომელზედაც რაღაცა სხვა ამოცანის ამოხსნით ამ საწყისი პრობლემის პასუხის დადგენა იქნება შესაძლებელი.
მაგალითად, მანქანებში ჩადგმული ნავიგაციის სისტემები, რომლელთა მეშვეობითაც ქალაქის ერთი ადგილიდან მეორეზე მისვლის უმოკლეს გზას ვგებულობთ, გრაფებზე ორ წვეროს შორის უმოკლესი გზის პოვნაზე დაიყვანება: თუ ქუჩების გადაკვეთას აღვნიშნავთ როგორც გრაფის წვეროებს, ხოლო წიბოებით კი თვითონ ქუჩებს, გამოგვივა შეწონილი გრაფი, რომელშიც ქუჩების სიგრძე წიბოს წონის ტოლი იქნება. ცხადია, რომ გრაფში უმოკლესი გზის პოვნა ქალაქში უმოკლესი გზის პოვნის ტოლფასი იქნება.
პირველ რიგში, აუცილებელია გადასაჭრელი ამოცანის მკაფიოდ და *სწორად* გადატანა გრაფებზე, რის შემდეგაც მის ამოსახსნელად რამოდენიმე ფუნდამენტური ალგორითმის ცოდნაა საჭირო, მათ შორის (მაგრამ არა მხოლოდ) უმცირესი დამფარავი ხის, მოცემული ორი წვეროს შორის უმოკლესი გზის, გრაფის პლანარულად (ბრტყლად) სიბრტყეზე დახაზვის ამოცანები. თუ ადამიანი რამოდენიმე ძირითადი ამოცანის გადაჭრის ხერხს დაეუფლება, უფრო რთული ამოცანების გადაჭრა, როგორც წესი, ამ ძირითადი ამოცანების თანმიმდევრულად გადაჭრის საშუალებითაც შეიძლება.
ამოცანათა გადაჭრის ძირითად მეთოდებს შორის (რომელიც ფართოდ გამოიყენება გრაფებზე ალგორითმებში) შეიძლება მოვიყვანოთ ე.წ. „სიგანეში ძებნის” და „სიღრმეში ძებნის” ალგორითმები, რომელთა საშუალებითაც სწრაფად შეგვიძლია შემოვიაროთ გრაფის ყველა წვერო (ყოველგვარი შეზღუდვის გარეშე).

ალგორითმებისა და მათი გადაჭრის მეთოდების შესწავლა უმჯობესია პრაქტიკული პრობლემების განხილვით დავიწყოთ.

1.2.1 ბერძნული მითი მინოტავრის შესახებ

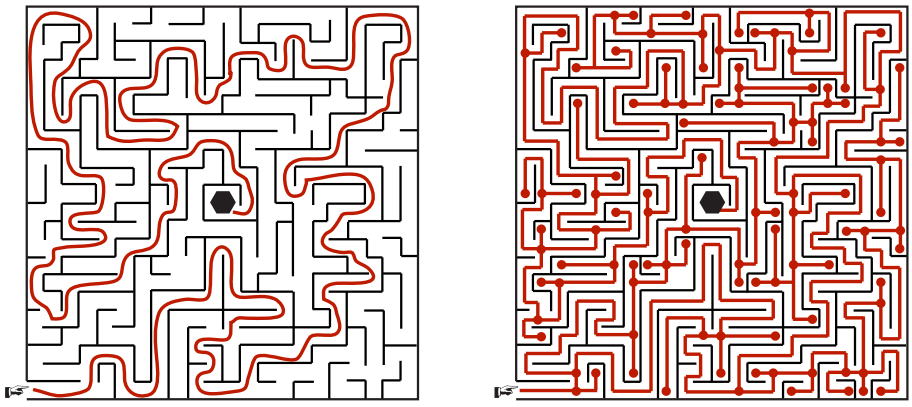
კუნძულ კრეტაზე ლაბირინთში დამწყვდეული იყო ადამიანის ტანისა და ხარის თავის მქონე ურჩხული მინოტავრი, რომლისთვის ათენელებს ხალხის მსხვერპლი უნდა შეეგზავნათ. ამ საშინელებისაგან თავის დახსნის მიზნით ბერძენი გმირი თესევსი ლაბირინთში შევიდა და მინოტავრი განგმირა. რადგან ლაბირინთში გზის გაკვლევა საკმაოდ რთული საქმეა, მან მეფის ქალიშვილის - არიადნას მიერ მიცემული ძაფი გამოიყენა, რითაც ადვილად გამოაგნო გარეთ (აქედან მომდინარეობს გამოთქმა „მსჯელობის ძაფი დაკარგა”, „ძაფი გაექცა” და სხვა).



ნახ. 1.1: მსოფლიოში ყველაზე განთქმული ლაბირინთი

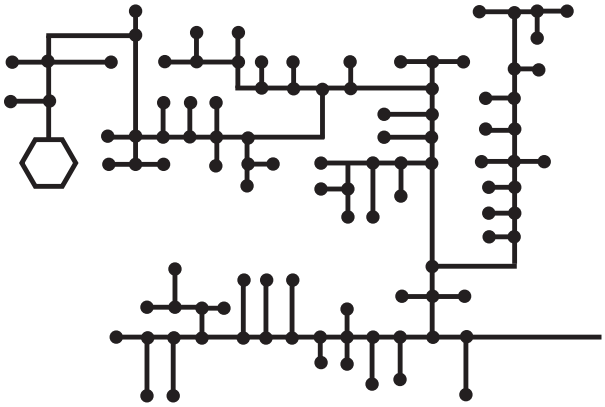
ეს მითი - ფილოსოფიური, ისტორიული, კულტურული და მრავალი სხვა მნიშვნელობის გარდა - ასევე დიდ როლს თამაშობს ინფორმატიკაშიც, რადგან უცნობ გარემოში მოძრაობის ამოცანას უკავშირდება, ხოლო ძაფის ან სხვა საშუალებებით განვლილი გზის მონიშნვა და უკან დაბრუნება ფართოდ გამოიყენება ალგორითმებთან დაკავშირებული პრობლემების გადასაჭრელად.

ყოველ ლაბირინთს შეგვიძლია შევუსაბამოთ რაღაცა გრაფი. ნახ. 1.2-ში ნაჩვენებია შედარებით რთული ლაბირინთი, რომელშიც შესაბამისი გრაფია ჩახაზული.



ნახ. 1.2: შედარებით რთული ლაბირინთი შესაბამისი გრაფით

იგივე გრაფი შეგვიძლია სხვანაირადაც დავხატოთ, რომ აღსაქმელად უფრო ადვილი იყოს (ნახ. 1.3). ცხადია, რომ თუ გვექნება ალგორითმი, რომლის საშუალებითაც გრაფის ყველა წვეროს შემოვლას შევძლებთ, ამით ლაბირინთში შესვლის ან გამოსვლის ალგორითმსაც ავაგებთ.



ნახ. 1.3: ლაბირინთის ექვივალენტური გრაფი

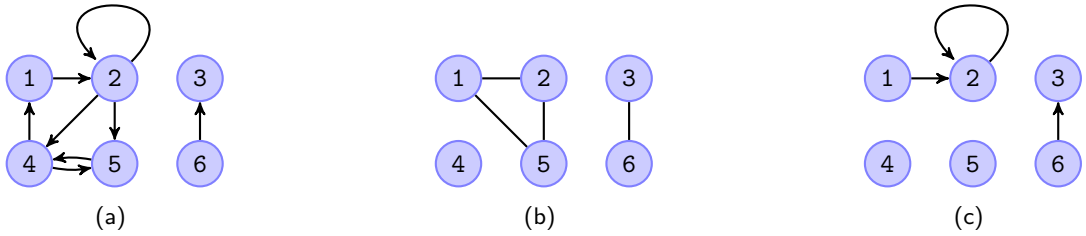
აღსანიშნავია, რომ ზედა ორ ნახაზში მოყვანილი გრაფი ერთმანეთის ექვივალენტურია (ერთის მეორეში გადაყვანა შეიძლება ისე, რომ გრაფის სტრუქტურა არ შეიცვალოს, აქ მხოლოდ დახატვის წესია სხვადასხვა), ამიტომ თუ ლაბირინთში შესვლისას გზის პირველ გასაყართან უნდა გავუხვიოთ მარცხნივ, მეორე გრაფში უნდა ვიაროთ პირდაპირ. მაგრამ ამას რაიმე პრინციპული მნიშვნელობა არ აქვს: ერთ გრაფზე მოძებნილი ამონახსნი ადვილად შეიძლება გადავიტანოთ მეორეზე.

ლაბირინთებში გზის გაკვლევის გარდა, გრაფში წვეროების შემოვლის ამოცანას მრავალი გამოყენება შეიძლება მოვუძებნოთ, მაგალითად, გრაფის წვეროების შიგთავსის ამობეჭდვაში, გრაფთა კოპირებასა ან სხვადასხვა ფორმატებში ჩაწერაში, წვეროთა ან წიბოთა რაოდენობის დათვლაში, გრაფის ბმული კომპონენტების პოვნაში, ორ წვეროს შორის გზის პოვნაში, გრაფში ციკლების აღმოჩენაში და ბევრ სხვა ამოცანაში.

თუ მოვახერხებთ იმას, რომ გრაფის შემოვლისას ყოველ წიბოზე გავივლით *ზუსტად ორჯერ* („იქით-აქეთ”), გვექნება იმის გარანტია, რომ არ გავიჭედებით და ყველა წვეროსაც გავივლით.

სავარჯიშო 1.1: ფორმალურად დაამტკიცეთ, რომ თუ გრაფის ყველა წიბოს შემოვუვლით ზუსტად ორჯერ, მის ყველა წვეროში ერთხელ მაინც შევალთ.

ახლა კი მოვიყვანოთ გრაფთა თეორიის ძირითადი განსაზღვრებები.
ორიენტირებული გრაფი (directed) G განისაზღვრება როგორც (V,E) წყვილი, სადაც V სასრული სიმრავლეა, ხოლო E წარმოადგენს V-ს ელემენტთა ბინარულ დამოკიდებულებას, ანუ $V \times V$ სიმრავლის ქვესიმრავლეა. ორიენტირებულ გრაფს ზოგჯერ **ორგრაფს** (digraph) უწოდებენ. V სიმრავლეს უწოდებენ გრაფის **წვეროთა სიმრავლეს** (vertex set), ხოლო E-ს - **წიბოთა სიმრავლეს** (edge set). მათ ელემენტებს შესაბამისად ეწოდებათ **წვერო** (vertex) და **წიბო** (edge). წიბოს, რომელიც წვეროს საკუთარ თავთან აერთებს, უწოდებენ **მარყუჟს** (**ციკლურ წიბოს**).
არაორიენტირებულ გრაფში (undirected graph) G=(V,E) წიბოთა E სიმრავლე შედგება წვეროთა დაულაგებელი (unordered) წყვილებისაგან. წიბოს აღსანიშნავად გამოიყენება ჩანაწერი (u,v). არაორიენტირებულ გრაფში (u,v) და (v,u) ერთი და იგივე წიბოს აღნიშნავს, ხოლო მარყუჟი არ შეიძლება არსებობდეს, რადგან წიბო ორი განსხვავებული წვეროსაგან უნდა შედგებოდეს.



ნახ. 1.4:

სურ. 1.4ა-ზე მოცემულია ორიენტირებული გრაფი 6 წვეროთი და 8 წიბოთი V={1, 2, 3, 4, 5, 6} და E={(1,2), (2,2), (2,4), (2,5), (4,1), (4,5), (5,4), (6,3)}. სურ. 1.4ბ-ზე - არაორიენტირებული გრაფი 6 წვეროთი და 4 წიბოთი V={1, 2, 3, 4, 5, 6} და E={(1,2), (1,5), (2,5), (3,6)}. (u,v) წიბოს შესახებ ორიენტირებულ გრაფში იტყვიან, რომ იგი გამოდის u წვეროდან და შედის v წვეროში. სურ. 1.4ა-ზე 2 წვეროდან გამოდის სამი წიბო - (2,2), (2,4), (2,5) და 2 წვეროში შედის ორი წიბო - (1,2), (2,2). არაორიენტირებულ გრაფში (u,v) წიბოს შესახებ იტყვიან, რომ იგი u და v წვეროების **ინციდენტურია** (incident).
თუ G გრაფში არსებობს (u,v) წიბო, იტყვიან, რომ v წვერო u წვეროს **მოსაზღვრეა** (is adjacent to u) არაორიენტირებულ გრაფებში მოსაზღვრეობა სიმეტრიული მიმართებაა, ხოლო ორიენტირებულ გრაფებისთვის ეს დებულება არ არის სამართლიანი. თუკი ორიენტირებულ გრაფში v წვერო u წვეროს მოსაზღვრეა, წერენ $u \to v$.
არაორიენტირებულ გრაფში წვეროს **ხარისხს** (degree) უწოდებენ ამ წვეროსადმი ინციდენტური წიბოების რაოდენობას. მაგალითად სურ. 1.4ბ-ზე 2 წვეროს ხარისხია 2. წვეროს, რომლის ხარისხიც არის 0, ეწოდება **იზოლირებული** (isolated) წვერო. წვეროს, რომლის ხარისხი არის 1 ეწოდება **დაკიდული** წვერო. ორიენტირებულ გრაფში განასხვავებენ **შემავალ** (in-degree) და **გამომავალ** (out-degree) ხარისხებს (შესაბამისად წვეროში შემავალი და გამომავალი წიბოების რაოდენობის მიხედვით) და მათ ჯამს უწოდებენ წვეროს ხარისხს. მაგალითად, სურ. 1.4ა-ზე 2 წვეროს ხარისხია 5 (შემავალი ხარისხი - 2, გამომავალი ხარისხი - 3). წვეროს, რომლის გამომავალი ხარისხი ნულია, ეწოდება **ჩასადენი** (sink), წვეროს, რომლის შემავალი ხარისხი ნულია, ეწოდება **წყარო** (source).
k **სიგრძის გზა** (**მარშრუტი**) (path of length k) u წვეროდან v წვეროში განიფაზღვრება როგორც წვეროთა $< v_0, v_1, v_2, \ldots, v_k >$ მიმდევრობა, სადაც $v_0 = u$, $v_k = v$ და $(v_{i-1}, v_i) \in E$ ნებისმიერი i=1,2,...,k-სათვის. გზის სიგრძე განისაზღვრება მასში შემავალი წიბოების რაოდენობით. გზა შეიცავს (contains) $v_0, v_1, v_2, \ldots, v_k$ წვეროებს და $(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$ წიბოებს. ყოველთვის არსებობს ნული სიგძის გზა წვეროდან თავის თავში. v_0 წვეროს უწოდებენ გზის **დასაწყისს**, ხოლო v_k წვეროს - გზის **ბოლოს** და ამბობენ, რომ გზა მიდის v_0-დან v_k-საკენ. თუ მოცემული u და u' წვეროებისთვის არსებობს p გზა u-დან u'-ში, მაშინ ამბობენ, რომ u' მიღწევადია u-დან p გზით (u' is reachable from u via p). გზას ეწოდება **მარტივი** (simple), თუკი ყველა წვერო მასში განსხვავებულია. სურ. 1.4ა-ზე 3 სიგრძის მქონე გზა ¡1, 2, 5, 4¿ მარტივია, ხოლო იმავე სიგრძის გზა ¡2, 5, 4, 5¿ - არაა მარტივი.
$p =< v_0, v_1, v_2, \ldots, v_k >$ გზის **ქვეგზა** (subpath) ეწოდება ამ გზიდან მიყოლებით აღებულ $< v_i, v_{i+1}, \ldots, v_j >$ წვეროთა მიმდევრობას, რომლისთვისაც $0 \leq i \leq j \leq k$. ორიენტირებულ გრაფში **ციკლი** (cycle) ეწოდება გზას, რომელშიც საწყისი წვერო ემთხვევა ბოლო წვეროს და რომელიც ერთ წიბოს მაინც შეიცავს. ციკლს ეწოდება **მარტივი**, თუკი მასში არ მეორდება არც ერთი წვერო პირველისა და ბოლოს გარდა. **მარყუჟი** წარმოადგენს ციკლს სიგრძით 1. აიგივებენ ციკლებს, რომლებიც განსხვავდებიან მხოლოდ წანაცვლებით ციკლის გასწვრივ. მაგ.: სურ. 1.4ა-ზე გზები ¡1, 2, 4, 1¿, ¡2, 4, 1, 2¿ და ¡4, 1, 2, 4¿ წარმოადგენენ ერთსა და იმავე ციკლს. ამავე ნახაზზე ციკლი ¡2, 2¿ შექმნილია ერთი წიბოთი. ორიენტირებულ გრაფს უწოდებენ **მარტივს**, თუკი იგი არ შეიცავს მარყუჟებს.

არაორიენტირებულ გრაფში $< v_0, v_1, v_2, \ldots, v_k >$ გზას ეწოდება **მარტივი ციკლი**, თუ $k \geq 3$, $v_0 = v_k$ და ყველა $v_1, v_2, \ldots, v_k$ წვერო განსხვავებულია. სურ. 1.4ბ-ზე მარტივი ციკლია ¡1, 2, 5, 1¿. გრაფს, რომელშიც არაა ციკლები, **აციკლური** (acyclic) ეწოდება.

არაორიენტირებულ გრაფს ეწოდება **ბმული** (connected), თუკი წვეროთა ნებისმიერი წყვილისათვის არსებობს გზა ერთიდან მეორეში. არაორიენტირებულ გრაფში გზის არსებობა ერთი წვეროდან მეორეში წარმოადგენს ეკვივალენტურ მიმართებას წვეროთა სიმრავლეზე. ეკვივალენტობის კლასებს ეწოდებათ გრაფის **ბმული კომპონენტები** (connected components). მაგ.: სურ. 1.4ბ-ზე სამი ბმული კომპონენტია: {1, 2, 5}, {3, 6} და {4}. არაორიენტირებული გრაფი ბმულია მაშინ და მხოლოდ მაშინ, როცა ის შედგება ერთადერთი ბმული კომპონენტისაგან.

ორიენტირებულ გრაფს ეწოდება **ძლიერად ბმული** (strongly connected), თუკი მისი ნებისმიერი წვეროდან მიღწევადია (ორიენტირებული გზებით) ნებისმიერი სხვა წვერო. ნებისმიერი ორიენტირებული გრაფი შეიძლება დაიყოს **ძლიერად ბმულ კომპონენტებად** (strongly connected components). სურ. 1.4ა-ზე არის სამი ასეთი კომპონენტი {1, 2, 4, 5}, {3} და {6}. შევნიშნოთ, რომ 3 და 6 წვეროები ერთად არ ჰქმნიან ძლიერად ბმულ კომპონენტს, რადგან არ არსებობს გზა 6-დან 3-ში.

G=(V,E) და G'=(V',E') გრაფებს ეწოდებათ **იზომორფულები** (isomorphic), თუკი არსებობს ურთიერთცალსახა შესაბამისობა $f : V \to V'$ მათი წვეროების სიმრავლეებს შორის, რომლის დროსაც $(u, v) \in E$ მაშინ და მხოლოდ მაშინ, როცა $(f(u), f(v)) \in E'$. შეიძლება ითქვას, რომ იზომორფული გრაფები ეს ერთი და იგივე გრაფია, სადაც წვეროები სხვადასხვაგვარადაა დასახელებული.

G'=(V',E') გრაფს ეწოდება G=(V,E) გრაფის **ქვეგრაფი** (subgraph), თუ $V' \subseteq V$ და $E' \subseteq E$. თუ G=(V,E) გრაფში ავირჩევთ V' წვეროთა ნებისმიერ სიმრავლეს, მაშინ შეგვიძლია განვიხილოთ G-ს ქვეგრაფი, რომელიც შედგება ამ წვეროებისა და მათი შემაერთებელი წიბოებისაგან. ამ ქვეგრაფს უწოდებენ G გრაფის **შეზღუდვას** V' წვეროთა სიმრავლეზე. სურ. 1.4ა-ზე მოცემული გრაფის შეზღუდვა {1, 2, 3, 6} წვეროთა სიმრავლეზე ნაჩვენებია სურ. 1.4ც-ზე და შეიცავს სამ წიბოს (1,2), (2,2), (6,3).

ნებისმიერი არაორიენტირებული გრაფისათვის შეიძლება განვიხილოთ მისი **ორიენტირებული ვარიანტი** (directed version), თუკი ყოველ (u,v) არაორიენტირებულ წიბოს შევცვლით ორიენტირებული წიბოების (u,v) და (v,u) წყვილით, რომლებსაც ექნებათ ურთიერთსაწინააღმდეგო მიმართულებები. მეორე მხრივ, ნებისმიერი ორიენტირებული გრაფისათვის შეიძლება განვიხილოთ მისი **არაორიენტირებული ვარიანტი** (undirected version), თუკი ამოვშლით მარყუჟებს და (u,v) და (v,u) წიბოებს შევცვლით არაორიენტირებული (u,v) წიბოთი. ორიენტირებულ გრაფში წვეროს **მეზობელი** (neighbor) ეწოდება ნებისმიერ წვეროს, რომელიც შეერთებულია მასთან ნებისმიერი მიმართულების წიბოთი, ე.ი. v წვერო არის u-ს მეზობელი თუ v არის u-ს მოსაზღვრე ან u არის v-ს მოსაზღვრე. არაორიენტირებულ გრაფში კი ცნებები "მეზობელი" და "მოსაზღვრე" სინონიმებია.

სრული (complete) გრაფი ეწოდება არაორიენტირებულ გრაფს, რომელიც შეიცავს ყველა შესაძლებელ წიბოს წვეროთა მოცემული სიმრავლისათვის, ე.ი. ნებისმიერი წვერო შეერთებულია ყველა დანარჩენთან. არაორიენტირებულ G = (V,E) გრაფს უწოდებენ **ორწილას** (bipartite), თუ V წვეროთა სიმრავლე შეიძლება გავყოთ ისეთ ორ V_1 და V_2 ნაწილად, რომ ნებისმიერი წიბოს ბოლოები სხვადასხვა ნაწილში აღმოჩნდეს. G=(V,E) გრაფის **დამატება** ეწოდება გრაფს, რომელსაც წვეროთა იგივე სიმრავლე აქვს, ხოლო ორი წვერო მოსაზღვრეა მაშინ და მხოლოდ მაშინ, როცა ისინი არ არიან მოსაზღვრე წვეროები G გრაფში. აციკლურ არაორიენტირებულ გრაფს უწოდებენ **ტყეს** (forest), ხოლო ბმულ აციკლურ არაორიენტირებულ გრაფს უწოდებენ (**თავისუფალ**) **ხეს** (free tree). **ორიენტირებული აციკლური გრაფის** (directed acyclic graph) აღსანიშნავად ზოგჯერ იყენებენ მის აბრევიატურას - DAG.

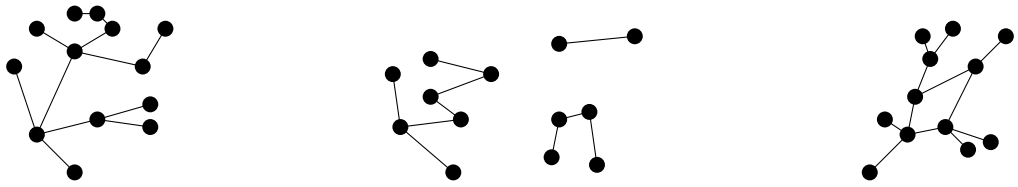
თუ გრაფის წვეროების ხარისხების საშუალო მნიშვნელობა $2|E|/|V|$ ახლოსაა $|V|$-სთან, ან, სხვა სიტყვებით რომ ვთქვათ, $|E|$ იმავე რიგისაა, რაც $|V|^2$, გრაფს ეწოდება **მკვრივი** (dence) გრაფი. **ხალვათი** (sparse) ეწოდება გრაფს, რომელშიც $|E|$ მკვეთრად მცირეა $|V|^2$-თან შედარებით. სხვა განმარტებით, **ხალვათი** ეწოდება გრაფს, რომლის დამატებაც მკვრივია.

1.3 ხეები

როგორც ზემოთ აღვნიშნეთ, ბმულ აციკლურ არაორიენტირებულ გრაფს უწოდებენ **ხეს** ან **თავისუფალ ხეს** (free tree), ხოლო აციკლურ არაორიენტირებულ გრაფს უწოდებენ ტყეს (forest). ტყე შედგება ხეებისაგან, რომლებიც მის ბმულ კომპონენტებს წარმოადგენენ. ხეებისათვის ვარგისი ბევრი ალგორითმი გამოსადეგია ტყეებისთვისაც. სურ. 1.5ა-ზე გამოსახულია ხე, სურ. 1.5ბ-ზე - ტყე (ის არ წარმოადგენს ხეს იმის გამო, რომ ბმული არაა), ხოლო სურ. 1.5ც მოცემული გრაფი არც ხეა და არც ტყე, რადგან იგი ციკლს შეიცავს.

თეორემა 1.1. (ხეთა თვისებები). ვთქვათ, G=(V,E) არაორიენტირებული გრაფია. მაშინ ტოლფასია შემდეგი თვისებები:

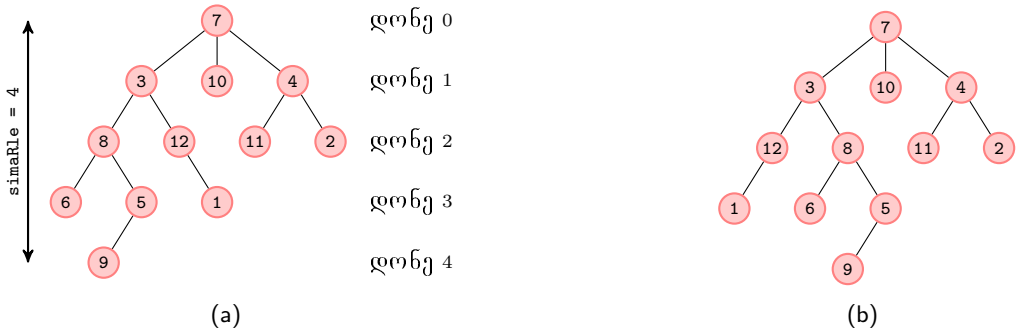
1. G ხეა



ნახ. 1.5:

2. G-ს ნებისმიერი ორი წვეროსათვის არსებობს მათი შემაერთებელი ერთადერთი მარტივი გზა
3. G გრაფი ბმულია, მაგრამ კარგავს ბმულობას, თუ ამოვიღებთ ნებისმიერ წიბოს
4. G გრაფი ბმულია და $|E| = |V| - 1$
5. G გრაფი აციკლურია და $|E| = |V| - 1$
6. G გრაფი აციკლურია, მაგრამ ნებისმიერი წიბოს დამატებით მასში ჩნდება ციკლი

ხე ფესვით (rooted tree) მიიღება მაშინ, როცა ბმულ აციკლურ არაორიენტირებულ გრაფში გამოყოფილია ერთ-ერთი წვერო, რომელსაც **ფესვს** (root) უწოდებენ. ხშირად, ფესვის მქონე ხის წვეროებს **კვანძებს** (nodes) უწოდებენ. სურ. 1.6-ზე ნაჩვენებია ფესვის მქონე ხე 12 წვეროთი და ფესვით 7.



ნახ. 1.6:

ვთქვათ x არის r ფესვის მქონე T ხის კვანძი. ნებისმიერ y კვანძს, რომელიც მდებარეობს (ერთადერთ) გზაზე r-დან x-ში, უწოდებენ x კვანძის **წინაპარს** (ancestor). თუ y არის x-ის წინაპარი, მაშინ x-ს უწოდებენ y-ის **შთამომავალს** (descendant). ყოველი კვანძი შეიძლება ჩაითვალოს საკუთარ წინაპრად ან შთამომავლად. თუ y არის x-ის წინაპარი და $x \neq y$, მაშინ y-ს უწოდებენ **საკუთარ წინაპარს** (proper ancestor), ხოლო x-ს უწოდებენ **საკუთარ შთამომავალს** (proper descendant).

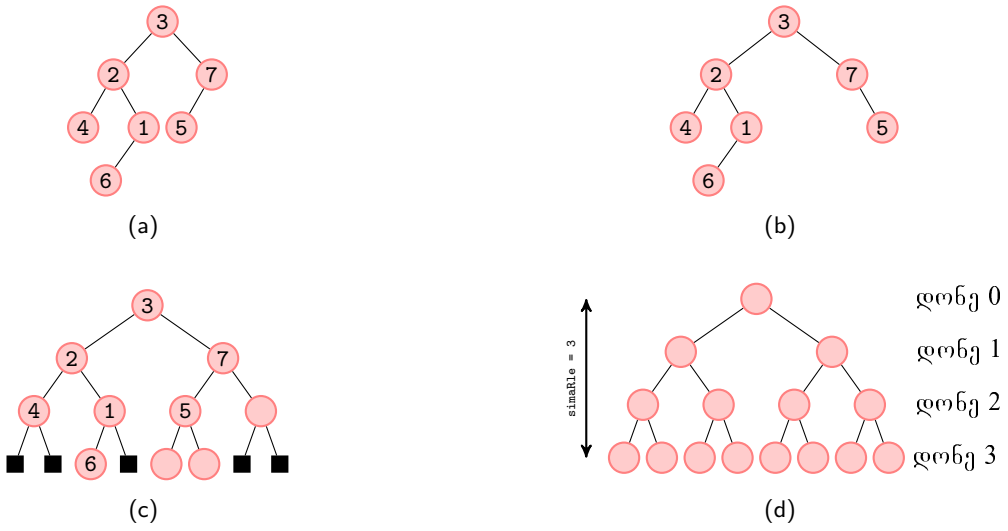
ყოველი x კვანძისათვის შეიძლება განვიხილოთ ხე, რომელიც x-ის ყველა შთამომავლისაგან შედგება და x ითვლება ამ ხის ფესვად. მას უწოდებენ **ქვეხეს** x **ფესვით**. მაგალითად სურ. 1.6ა-ზე ქვეხე ფესვით 8 შეიცავს წვეროებს 8, 6, 5 და 9.

თუ (y,x) უკანასკნელი წიბოა T ხეში r ფესვიდან x-საკენ მიმავალ გზაზე, მაშინ y-ს უწოდებენ x-ის **მშობელს** (parent), ხოლო x-ს უწოდებენ y-ის **შვილს** (child). ფესვი ერთადერთი კვანძია, რომელსაც მშობელი არ ჰყავს. კვანძებს, რომელთაც საერთო მშობელი ჰყავს, უწოდებენ **დედმამიშვილებს** (siblings, რუსულ ლიტერატურაში - ძმები). ფესვის მქონე ხის კვანძს, რომელსაც არა ჰყავს შვილები, უწოდებენ **ფოთოლს** (leaf, external node). წვეროებს, რომელთაც ჰყავთ შვილები, უწოდებენ **შინაგანს** (internal). ფესვის მქონე ხის კვანძის შვილების რაოდენობას უწოდებენ მის **ხარისხს** (degree). გზის სიგრძეს ფესვიდან ნებისმიერ x კვანძამდე უწოდებენ x წვეროს **დონეს** (depth). ხის კვანძის **სიმაღლე** (height) არის წიბოების რაოდენობა ყველაზე გრძელ გზაზე ამ კვანძიდან მის შთამომავალ ფოთლამდე. ხის სიგრძედ ითვლება მისი ფესვის სიგრძე.

დალაგებული ხე (ordered tree) ეწოდება ფესვის მქონე ხეს, რომელსაც დამატებითი სტრუქტურა აქვს: ყოველი კვანძისათვის მისი შვილების სიმრავლე დალაგებულია (ცნობილია თუ რომელია წვეროს პირველი შვილი, მეორე შვილი და ა.შ.). სურ. 1.6-ზე გამოსახული ორ ხეს ერთი და იგივე ფესვი აქვს და ისინი განსხვავდებიან მხოლოდ შვილების დალაგებით.

ორობითი ხე (binary tree) ყველაზე მარტივი სახით განისაზღვრება რეკურსიულად, როგორც კვანძთა სასრული სიმრავლე, რომელიც: ან ცარიელია (არ შეიცავს კვანძებს), ან გაყოფილია სამ არაგადამკვეთ ნაწილად: წვერო

რომელსაც ეწოდება **ფესვი** (root), ორობითი ხე, რომელსაც ეწოდება ფესვის **მარცხენა ქვეხე** (left subtree) და ორობითი ხე, რომელსაც ეწოდება ფესვის **მარჯვენა ქვეხე** (right subtree) ორობით ხეს, რომელიც არ შეიცავს კვანძებს, ეწოდება **ცარიელი** (empty). ზოგჯერ მას აღნიშნავენ NIL-ით. თუ მარცხენა ქვეხე არაცარიელია, მაშინ მის ფესვს უწოდებენ მთლიანი ხის ფესვის **მარცხენა შვილს** (left child), შესაბამისად განისაზღვრება **მარჯვენა შვილიც** (right child). ორობითი ხის მაგალითი მოცემულია სურ. 1.7-ზე.
არასწორი იქნებოდა განგვესაზღვრა ორობითი ხე, როგორც დალაგებული ხე, რომელშიც თითოეული კვანძის ხარისხი არ აღემატება 2-ს. ამის მიზეზია ის, რომ ორობით ხეში მნიშვნელობა აქვს როგორია კვანძის ერთადერთი შვილი - მარცხენა თუ მარჯვენა, ხოლო დალაგებული ხისათვის ასეთი განსხვავება არ არსებობს. სურ. 1.7ა-ზე და სურ. 1.7ბ-ზე ნაჩვენები ორობითი ხეები განსხვავდებიან, რადგან 1.7ა-ზე 5 არის 7-ის მარცხენა შვილი, ხოლო 1.7ბ-ზე - მარჯვენა. როგორც დალაგებული ხეები ისინი ერთნაირები არიან.



ნახ. 1.7:

ხშირად ორობით ხეზე ცარიელ ადგილებს ავსებენ ფიქტიური ფოთლებით, მიიღება **სრულად ორობითი ხე** (full binary tree). ამის შემდეგ ყველა კვანძი ან ფოთოლია, ან აქვს ხარისხი 2, 1-ს ტოლი ხარისხის მქონე კვანძები ასეთ ხეში არ არის. ეს გარდაქმნა ნაჩვენებია სურ. 1.7ც-ზე.
სრული k -ობითი ხე (complete k-ary tree) ეწოდება k-ობით ხეს, რომელშიც ყველა ფოთოლს აქვს ერთნაირი დონე და ყველა შინაგან კვანძს აქვს ხარისხი k. ასეთ შემთხვევაში ხის სტრუქტურა მთლიანად განისაზღვრება მისი სიმაღლით. სურ. 1.7დ-ზე გამოსახულია სრული ორობითი ხე სიმაღლით 3. ფესვი ყოველთვის არის 0 დონის ერთადერთი კვანძი, მისი k შვილი არის 1 დონის მქონე კვანძები, რომელთა k^2 შვილი წარმოადგენენ 2 დონის კვანძებს და ა.შ. h დონეზე გვექნება k^h ფოთოლი. h სიმაღლის სრული k-ობითი ხის შინაგანი კვანძების რაოდენობა ტოლია:

$$1 + k + k^2 + \cdots + k^{h-1} = \frac{k^h - 1}{k - 1}$$

კერძოდ, სრული ორობითი ხის შინაგანი კვანძების რაოდენობაა 2^h-1.

1.4 გრაფის წარმოდგენა

გრაფის წარმოდგენისთვის გამოიყენება სამი ძირითადი სტრუქტურა:

1. **მოსაზღვრე წვეროების სია** (adjacency list representation)
2. **მოსაზღვრეობის მატრიცა** (adjacency matrix representation)
3. **წიბოების სია** (edge list representation)

G=(V,E) გრაფის წარმოდგენა მოსაზღვრე წვეროების სიებით იყენებს V ცალი წვეროსგან შემდგარ ბმულ სიას. მოსაზღვრე წვეროთა ყველა სიის სიგრძეთა ჯამი ორიენტირებული გრაფისათვის წიბოთა რაოდენობის ტოლია, ხოლო არაორიენტირებული გრაფისათვის - წიბოთა გაორმაგებული რაოდენობის ტოლი, რადგან (u,v) წიბო წარმოშობს ელემენტს როგორც u, ასევე v წვეროს მოსაზღვრე წვეროთა სიაში. ორივე შემთხვევაში მეხსიერების

საჭირო მოცულობაა O(V+E). ამ წარმოდგენით მოსახერხებელია წონადი გრაფების (weighted graphs) შენახვაც, სადაც ყოველ წიბოს შეესაბამება ნამდვილი რიცხვი - ამ წიბოს **წონა** (weight), ანუ მოცემულია **წონის ფუნქცია** (weight function) $w : E \to R$. ამ შემთხვევაში მოსახერხებელია $(u, v) \in E$ წიბოს w(u,v) წონის შენახვა u წვეროსთან ერთად წვეროს მოსაზღვრე წვეროთა სიაში. თუმცა ამ წარმოდგენას აქვს მნიშვნელოვანი ნაკლი: u-დან v-ში წიბოს არსებობის დასადგენად, საჭიროა გადავამოწმოთ მთლიანი სია, მასში v-ს მოსაძებნად. ძებნას შესაძლოა თავი ავარიდოთ, თუ გამოვიყენებთ მოსაზღვრეობის მატრიცას, თუმცა ამ შემთხვევაში მეტი მანქანური მეხსიერებაა საჭირო.
მოსაზღვრეობის მატრიცის გამოყენებისას უნდა გადავნომროთ G=(V,E) გრაფის წვეროები 1,2,. . . ,V რიცხვებით და განვიხილოთ $|V| \times |V|$ ზომის $A = (a_{ij})$ მატრიცა, სადაც $a_{ij} = 1$, თუ $(i, j) \in E$ და $a_{ij} = 0$, წინააღმდეგ შემთხვევაში. მეხსიერების საჭირო მოცულობაა $O(V^2)$. არაორიენტირებული გრაფისათვის მოსაზღვრეობის მატრიცა სიმეტრიულია მთავარი დიაგონალის მიმართ და ამიტომ საკმარისია შევინახოთ მხოლოდ მთავარი დიაგონალი და მის ზემოთ მყოფი ელემენტები, რაც თითქმის ორჯერ ამცირებს გამოყენებულ მეხსიერებას. წონადი გრაფების შენახვა პრობლემა არც ამ მეთოდისთვისაა - (u,v) წიბოს w(u,v) წონა მატრიცაში თავსდება u სტრიქონისა და v სვეტის გადაკვეთაზე, ხოლო წიბოს არარსებობის შემთხვევაში მატრიცის შესაბამისი ელემენტი მიიღებს ცარიელ მნიშვნელობას NIL (ზოგ ამოცანაში შეიძლება გამოვიყენოთ 0 ან ∞).
სურ. 1.8-ზე მოცემულია გრაფთა წარმოდგენის ორივე მეთოდი როგორც არაორიენტირებული, ისევე ორიენტირებული გრაფებისათვის.
თითოეული წარმოდგენის გამოყენების შესახებ გადაწყვეტილების მიღება დამოკიდებულია: გრაფის განზომილებაზე (მოსაზღვრეობის მატრიცით წარმოდგენა მოსახერხებელია პატარა ან მკვრივი გრაფებისათვის), ალგორითმზე (მოსაზღვრეობის მატრიცით წარმოდგენა უმჯობესია ალგორითმებისთვის, რომლებიც პასუხობენ კითხვაზე, მაგ.: არის თუ არა (u,v) წიბო G გრაფში?) იმაზე, ხალვათია თუ მკვრივი გრაფი (თუ გრაფი მკვრივია, მატრიცით წარმოდგენა უფრო მოსახერხებელია, რადგან ყველა შემთხვევაში $O(V^2)$ მეხსიერების გამოყენება მოგვიწევს).
თუკი მანქანური მეხსიერება საკმარისია, სჯობს გრაფი აღვწეროთ მოსაზღვრეობის მატრიცის საშუალებით, რადგან უმეტეს შემთხვევაში მასთან მუშაობა უფრო მოსახერხებელია. ამას გარდა, თუკი გრაფი წონადი არ არის, მოსაზღვრეობის მატრიცის ელემენტები შესაძლოა განიხილოთ როგორც ბიტები, რომლებიც შეგვიძლია გავაერთიანოთ მანქანურ სიტყვებში - ეს იძლევა მეხსიერების მნიშვნელოვან ეკონომიას.
გრაფისთვის ეფექტური ალგორითმის შერჩევის დროს, ერთ-ერთი მთავარი ფაქტორია ინფორმაცია იმის შესახებ ხალვათია თუ მკვრივი გრაფი. მაგ.: თუ რაიმე ამოცანის გადასაწყვეტად შეგვიძლია შევიმუშაოთ 2 ალგორითმი: პირველს სჭირდება —V—^{2}, ხოლო მეორეს - $|E| \log |E|$ ბიჯი. ეს ფორმულები გვიჩვენებს, რომ პირველი ალგორითმი უფრო გამოდგება მკვრივი გრაფებისთვის, ხოლო მეორე - ხალვათისთვის. მაგ.: განვიხილოთ მკვრივი გრაფი —E—=10^6 და —V—=10^3, —V—^{2} 20-ჯერ ჩქარია $|E| \log |E|$-ზე. მეორე მხრივ, ხალვათი გრაფისთვის, —E—=10^6 და —V—=10^6 ალგორითმი $|E| \log |E|$ სირთულით მილიონის რიგით აჯობებს —V—^{2} სირთულის ალგორითმს.

1.5 სავარჯიშოები

1. ოთხი ადამიანი ღამით მოძრაობს გზაზე ერთი მიმართულებით. მათ უნდა გადაიარონ ხიდზე და აქვთ მხოლოდ 1 ფარანი. ხიდზე ერთდროულად შეუძლია გაიაროს არაუმეტეს 2 ადამიანმა (ან ერთმა, ან ორმა) და ერთ-ერთს აუცილებლად უნდა ეჭიროს ფარანი. არ შეიძლება ფარანის ერთი ნაპირიდან მეორეზე გადაგდება, შეიძლება მისი მხოლოდ ხიდით გადატანა. თითოეული ადამიანი ხიდის გადასვლას უნდება: პირველი-1წთ, მეორე-2წთ, მესამე-5წთ, მეოთხე-10წთ. თუ ხიდზე გადადის 2 ადამიანი, ისინი მოძრაობენ უფრო ნელი სიჩქარით. რა თანმიმდევრობით უნდა გადაიარონ ხიდი, რომ ამისთვის დასჭირდეთ ზუსტად 17 წუთი.

2. მოიყვანეთ შემდეგი გრაფების მაგალითები ან აჩვენეთ, რომ ასეთი გრაფები არ არსებობს:

   (ა) გრაფი, რომელსაც აქვს ჰამილტონის ციკლი, მაგრამ არ აქვს ეილერის ციკლი

   (ბ) გრაფი, რომელსაც აქვს ეილერის ციკლი, მაგრამ არ აქვს ჰამილტონის ციკლი

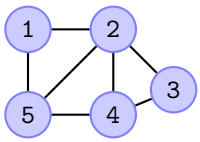
   (ც) გრაფი, რომელსაც აქვს როგორც ეილერის, ისე ჰამილტონის ციკლი

   (დ) გრაფი, რომელსაც აქვს ყველა წვეროს შემცველი ციკლი, მაგრამ არ აქვს არც ჰამილტონის, არც ეილერის ციკლი

3. დაამტკიცეთ, რომ V წვეროს შემცველი არაორიენტირებული გრაფი შეიცავს არაუმეტეს $|V|(|V|-1)/2$ წიბოს.

4. დაამტკიცეთ, რომ ნებისმიერი G=(V,E) არაორიენტირებული ბმული გრაფისთვის სრულდება $|E| \geq |V| - 1$.

5. დაამტკიცეთ, რომ ნებისმიერი G=(V,E) აციკლური ბმული არაორიენტირებული გრაფი შეიცავს $|V| - 1$ წიბოს.

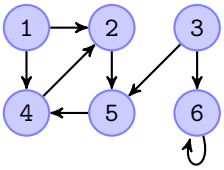


1	2, 5
2	1, 3, 4, 5
3	2, 4
4	2, 3, 5
5	1, 2, 4

(a) მოსაზღვრე წვეროების სია

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(b) მოსაზღვრეობის მატრიცა



1	2, 4
2	5
3	5, 6
4	2
5	4
6	6

(c) მოსაზღვრე წვეროების სია

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

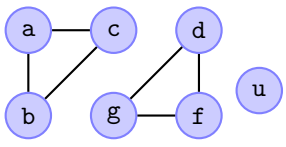
(d) მოსაზღვრეობის მატრიცა

ნახ. 1.8:

6. (ა) გრაფის მოსაზღვრეობის მატრიცის რა თვისებები მეტყველებს იმაზე, რომ:
   i. გრაფი სრულია
   ii. გრაფი შეიცავს მარყუჟს
   iii. გრაფი შეიცავს იზოლირებულ წვეროს

   (ბ) გაეცით პასუხები (ა)-ში დასმულ შეკითხვებზე გრაფის მოსაზღვრე წვეროთა სიით წარმოდგენის შემთხვევისთვის.

7. იპოვეთ ეკვივალენტობის კლასები შემდეგი გრაფისათვის:

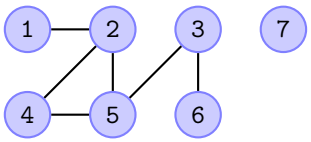


8. შეამოწმეთ, რომ მიმართება "მიღწევადია -დან" არაორიენტირებულ გრაფში არის ეკვივალენტობის მიმართება გრაფის წვეროთა სიმრავლეზე.

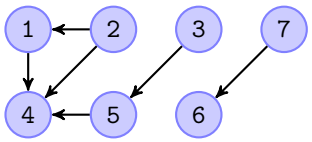
9. ეკვივალენტობის მიმართების 3 თვისებიდან რომელი ირღვევა "მიღწევადია -დან" მიმართებისთვის ორიენტირებულ გრაფში?

10. რამდენი ბმული კომპონენტი აქვს ხეს?

11. შემდეგი გრაფებისთვის ააგეთ მოსაზღვრეობის მატრიცა და მოსაზღვრე წვეროთა სია.



(a)



(b)

თავი 2

გრაფის შემოვლის ალგორითმები

ბევრი ალგორითმი, რომელიც მუშაობს გრაფებზე, საჭიროებს გრაფის წვეროების და წიბოების გარკვეულ დამუშავებას. გრაფების შემოვლისთვის არსებობს ორი ძირითადი ალგორითმი - **სიგანეში ძებნა** (breadth-first search, BFS) და **სიღრმეში ძებნა** (depth-first search, DFS). ეს ალგორითმები გამოიყენება როგორც არაორიენტირებული, ისე ორიენტირებული გრაფებისთვის. გარდა მათი ძირითადი დანიშნულებისა (წვეროებისა და წიბოების შემოვლა), მათი გამოყენება სასარგებლოა გრაფების რიგი მნიშვნელოვანი თვისების დასადგენად (მაგ, გრაფის ბმულობის, აციკლურობის და ა.შ.)

2.1 სიგანეში ძებნის ალგორითმი

სიგანეში ძებნის ალგორითმი გრაფის შემოვლის ერთ-ერთი მარტივი ალგორითმია, რომელიც გრაფებთან მომუშავე ბევრი ალგორითმის საფუძველს წარმოადგენს. მაგალითად, მინიმალური დამფარავი ხის აგების პრიმის ალგორითმი, ერთი წვეროდან უმოკლესი მანძილის პოვნის დეიქსტრას ალგორითმი იყენებენ სიგანეში ძებნის ალგორითმის მსგავს იდეებს.
ვთქვათ მოცემულია G=(V,E) გრაფი და მისი s **საწყისი წვერო** (source vertex). სიგანეში ძებნის (breadth-first search) ალგორითმი ადგენს უმოკლეს მანძილს s-დან ყველა მიღწევად წვერომდე (მანძილად აქ ითვლება უმოკლეს გზაზე წიბოთა რაოდენობა, წიბოს წონა ერთის ტოლად ითვლება). ალგორითმის მუშაობის პროცესში მიიღება ე.წ. ძებნის ხე, რომელიც მოიცავს ფესვიდან მიღწევად ყველა წვეროს.
მეტი თვალსაჩინოებისათვის ჩავთვალოთ, რომ ალგორითმის მუშაობის პროცესში გრაფის წვეროები შესაძლოა იყოს თეთრი, რუხი ან შავი ფერის. თავდაპირველად ყველა წვერო თეთრი ფერისაა, ხოლო ალგორითმის მუშაობის დროს თეთრი წვერო შეიძლება გადაიქცეს რუხად, ხოლო რუხი - შავად (მაგრამ არა პირიქით). რუხად იღებება ის წვერო, რომელიც ალგორითმმა აღმოაჩინა, მაგრამ ჯერ არაა შემოწმებული მისგან გამომავალი სხვა წიბოები, ხოლო შავად იღებება ის წვერო, რომლისგანაც გამომავალი ყველა წიბო უკვე შემოწმებულია.
თავიდან ძებნის ხე შედგება მხოლოდ ფესვისაგან. როცა ალგორითმი პირველად აღმოაჩენს ახალ (თეთრი ფერის) v წვეროს, რომელიც უკვე ნაპოვნი u წვეროს მოსაზღვრეა, v წვერო ხდება u წვეროს **შვილი** (child) და იღებება რუხად, ხოლო (u,v) წიბო ემატება ძებნის ხეს. ასეთ წიბოს უწოდებენ **ხის წიბოს** (tree edge). u წვერო ხდება v წვეროს **მშობელი** (parent) და თუკი მისი ყველა შვილი ნაპოვნია, იღებება შავად. თუ წიბოს მივყავართ უკვე აღმოჩენილ წვეროსთან, რომელიც არ წარმოადგენს მის უშუალო წინაპარს, მას უწოდებენ **ჯვარედინ წიბოს** (cross edge). ყოველი წვეროს აღმოჩენა ხდება მხოლოდ ერთხელ, ამიტომ წვეროს არ შეიძლება ერთზე მეტი მშობელი ჰყავდეს. "წინაპრისა" და "შთამომავლის" ცნებებიც ამ შემთხვევაში ჩვეულებრივად განისაზღვრება.
პროცედურა BFS (breadth-first-search - განივად ძებნა) იყენებს გრაფის წარმოდგენას მოსაზღვრე წვეროთა სიით. ყოველი u წვეროსათვის დამატებით ინახება მისი ფერი color[u] და მისი მშობელი π[u]. თუკი მშობელი ჯერ ნაპოვნი არ არის ან u=s, მაშინ π[u]=NIL. მანძილი s-დან u-მდე იწერება d[u] ველში. რუხი წვეროების შესანახად გამოიყენება რიგი Q (განმარტება თავის ბოლოს).
2-5 სტრიქონებში ყველა წვეროსათვის ფერი ხდება თეთრი, მნიშვნელობები - უსასრულობა, ხოლო მშობლები - NIL. მე-6 სტრიქონში ხდება ფესვის (s წვეროს) დამუშავება. მე-7 სტრიქონში Q ცარიელ რიგს ემატება მისი პირველი წევრი - s წვერო. პროგრამის ძირითადი ციკლი (8-16 სტრიქონები) სრულდება Q რიგის დაცარიელებამდე, ე.ი. სანამ არსებობენ რუხი ფერის წვეროები, ანუ წვეროები, რომლებიც აღმოჩენილია, მაგრამ რომელთა მოსაზღვრეობის სიები ჯერ განხილული არ არის. პირველი იტერაციის წინ, ერთადერთი რუხი ფერის წვერო და ერთადერთი წვერო არის საწყისი s წვერო. მე-9 სტრიქონით განისაზღვრება რუხი წვერო u რიგის თავში, რომელიც შემდეგ ამოიშლება Q რიგიდან. 10-15 სტრიქონებში for ციკლი განიხილავს u-ს ყველა მოსაზღვრე v წვეროს მოსაზღვრე წვეროთა სიაში. თუკი მათ შორის აღმოჩნდება თეთრი ფერის წვერო, ის იღებება რუხად, მის

Algorithm 1: Breadth First Search (BFS)

Input: გრაფი $G = (V, E)$ და საწყისი წვერო s
Output: გრაფის განივად ძებნისას აღმოჩენილი წვეროების მიმდევრობა

```
1  BFS(G, s) :
2      for ∀u ∈ V\{s} :
3          color[u] = TeTri;
4          d[u] = ∞;
5          π[u] = NIL;
6      color[s] = ruxi; d[s] = 0; π[s] = NIL; Q = ∅;
7      ENQUEUE(Q, s);
8      while Q ≠ ∅ :
9          u = DEQUEUE(Q);
10         for ∀v ∈ Adj[u] :
11             if color[v] == TeTri :
12                 color[v] = ruxi;
13                 d[v] = d[u] + 1;
14                 π[v] = u;
15                 ENQUEUE(Q, v);
16         color[u] = Savi;
17     return d, π
```

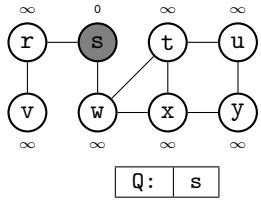
მშობლად ცხადდება u და მანძილად ფესვამდე - d[u]+1, ხოლო თავად ეს წვერო თავსდება Q რიგის ბოლოში. ამის შემდეგ u წვერო იღებება შავად (16 სტრიქონი).
სურ. 2.1-ზე მოცემულია BFS პროცედურის მუშაობა არაორიენტირებული გრაფისათვის და აგებულია სიგანეში ძებნის ხე. მუქად აღნიშნულია **ხის** წიბოები T (tree edges). წიბოები, რომლებიც არ შედიან სიგანეში ძებნის ხეში, არის **ჯვარედინი** წიბოები - C (cross edges).
სიგანეში ძებნის შედეგი შეიძლება დამოკიდებული იყოს u-ს მოსაზღვრე წვეროების თანმიმდევრობაზე, მე-10 სტრიქონში. სიგანეში ძებნის ხეები შეიძლება განსხვავდებოდეს, მაგრამ მანძილი d, რომელსაც ითვლის ალგორითმი, არ არის დამოკიდებული წვეროთა განხილვის რიგზე.
განვსაზღვროთ პროცედურის მუშაობის დრო. ყოველი წვერო რიგში თავსდება მხოლოდ ერთხელ და ასევე ერთხელ ხდება მისი ამოღება რიგიდან, ამიტომ რიგთან დაკავშირებულ ოპერაციებზე დაიხარჯება O(V) დრო. მოსაზღვრე წვეროთა სია ასევე ერთხელ განიხილება, როცა შესაბამისი წვერო ამოიღება რიგიდან. მოსაზღვრე წვეროთა სიებში ელემენტთა ჯამური სიგრძე კი E-ს ტოლია (არაორიენტირებულ გრაფში $2|E|$), ამიტომ ამ ოპერაციაზე დაიხარჯება O(E) დრო. ინიციალიზაციას სჭირდება O(V) დრო. მაშასადამე, ალგორითმის მუშაობის საერთო დრო იქნება O(V+E).
სიგანეში ძებნა პოულობს მანძილებს ფესვიდან გრაფის თითოეულ მიღწევად წვერომდე. განვსაზღვროთ **უმოკლესი გზის სიგრძე** (shortest-path distance) δ(s,v) - s ფესვიდან v წვერომდე, როგორც წიბოების მინიმალური რაოდენობა s ფესვიდან v წვერომდე რაიმე გზაზე. თუ გზა s-დან v-მდე არ არსებობს, მაშინ $\delta(s, v) = \infty$. $\delta(s, v)$ სიგრძის გზას s ფესვიდან v წვერომდე ეწოდება **უმოკლესი გზა** (shortest path). ასეთი გზა შეიძლება რამდენიმე იყოს. ქვემოთ განხილული იქნება უმოკლესი გზების უფრო ზოგადი სახე, როცა საქმე გვაქვს წონიან წიბოებთან და გზის სიგრძე წიბოების წონათა ჯამის ტოლია. ჩვენს შემთხვევაში კი წიბოთა წონები ერთეულის ტოლად ითვლება და გზის სიგრძე წიბოთა რაოდენობას უდრის.

ლემა 2.1. ვთქვათ მოცემულია G=(V,E) გრაფი (ორიენტირებული ან არაორიენტირებული) და მისი ნებისმიერი s წვერო. მაშინ მისი ნებისმიერი $(u, v) \in E$ წიბოსთვის სამართლიანია $\delta(s, v) \leq \delta(s, u) + 1$.

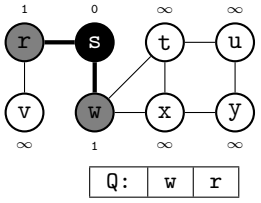
Proof. თუ u წვერო მიღწევადია s-დან, მაშინ მიღწევადია v-ც. ამ შემთხვევაში, უმოკლესი გზა s-დან v-დე არ შეიძლება იყოს იმაზე გრძელი, ვიდრე უმოკლესი გზა s-დან u-მდე, რომელსაც მოსდევს (u,v) წიბო. ასე, რომ დასამტკიცებელი უტოლობა სრულდება. ხოლო, თუ u წვერო არ არის მიღწევადი s-დან, მაშინ, $\delta(s, u) = \infty$ და უტოლობა ამ შემთხვევაშიც სრულდება. □

ლემა 2.2. ვთქვათ მოცემულია G=(V,E) გრაფი (ორიენტირებული ან არაორიენტირებული) და ვთქვათ, სრულდება პროცედურა BFS(G,s), მაშინ პროცედურის დასრულების შემდეგ, ყოველი $v \in V$ წვეროსთვის, მნიშვნელობა d[v], გამოთვლილი BFS(G,s) პროცედურით, აკმაყოფილებს უტოლობას $d[v] \geq \delta(s, v)$.

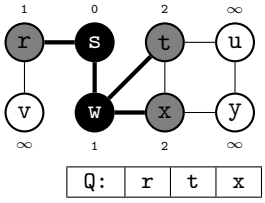
Proof. გამოვიყენოთ ინდუქცია ENQUEUE ოპერაციის რაოდენობის მიხედვით. ინდუქციის ჰიპოთეზა მდგომარეობს იმაში, რომ ყოველი $v \in V$ წვეროსთვის, სრულდება პირობა $d[v] \geq \delta(s, v)$.



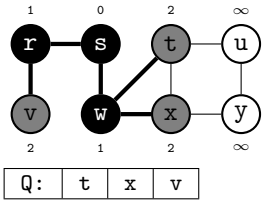
(a)



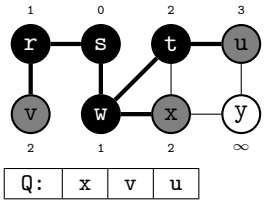
(b)



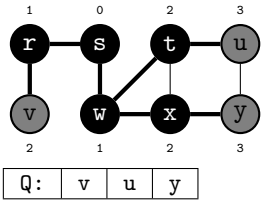
(c)



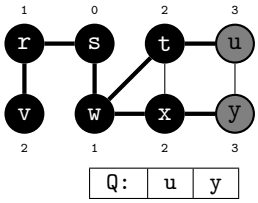
(d)



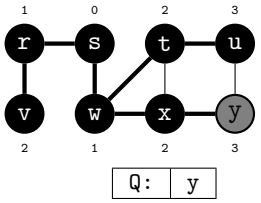
(e)



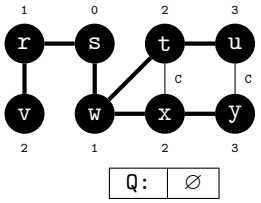
(f)



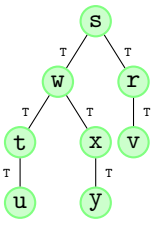
(g)



(h)



(i)



(j)

ნახ. 2.1:

ინდუქციის ბაზისი არის მდგომარეობა, რომელიც დგება s საწყისი წვეროს რიგში დამატების შემდეგ BFS(G,s) პროცედურის მე-7 სტრიქონში. ამ შემთხვევაში პირობა სრულდება: $d[s] = 0 = \delta(s, s)$, ხოლო $\forall v \in V - \{s\}$-სთვის $d[v] = \infty \geq \delta(s, v)$.
ინდუქციის ყოველ ბიჯზე განვიხილოთ თეთრი წვერო v, რომელიც იხსნება ძებნის პროცესში u წვეროდან. ინდუქციის ჰიპოთეზის თანახმად, $d[u] \geq \delta(s, u)$. მე-13 სტრიქონში მინიჭებისა და ლემა 2.1-ს გამოყენებით, მივიღებთ:

$$d[v] = d[u] + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$$

ამის შემდეგ, v წვერო ემატება რიგს. რადგან იგი ამ დროს იღებება რუხად, ხოლო 12-15 სტრიქონები სრულდება მხოლოდ თეთრი წვეროებისთვის, v წვერო რიგს აღარ დაემატება, ამრიგად, მისი მნიშვნელობა d[v] აღარ შეიცვლება, ასე რომ ინდუქციის ჰიპოთეზა სრულდება. □

ლემა 2.3. ვთქვათ, BFS(G,s) პროცედურის შესრულების პროცესში, Q რიგში შედის წვეროები $< v_1, v_2, \ldots, v_r >$, სადაც v_1 - Q რიგის თავია, ხოლო v_r - Q რიგის ბოლო. მაშინ ყოველი i=1, 2, ..., r-1-სთვის სამართლიანია $d[v_r] \leq d[v_1] + 1$ და $d[v_i] \leq d[v_{i+1}]$.

Proof. დავამტკიცოთ ინდუქციით, Q რიგთან ჩატარებული ოპერაციების რაოდენობის მიხედვით. ინდუქციის ბაზისად ჩავთვალოთ მდგომარეობა, როცა რიგში არის ერთი s წვერო. ამ შემთხვევაში ლემის პირობა სრულდება. ინდუქციის ყოველ ბიჯზე უნდა დავამტკიცოთ, რომ ლემა სრულდება Q რიგში წვეროს როგორც ჩამატების, ისე ამოღების შემდეგაც.
თუ რიგიდან ვიღებთ მის თავს, v_1-ს, რიგის ახალი თავი ხდება v_2 (თუ რიგი ცარიელდება რაიმე წვეროს რიგიდან ამოღებით, ლემა სრულდება). ინდუქციის ჰიპოთეზის თანახმად, $d[v_1] \leq d[v_2]$, მაგრამ მაშინ მივიღებთ $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$, ხოლო ყველა დანარჩენი უტოლობა რჩება უცვლელი. ამრიგად, ლემა სრულდება, როცა რიგის ახალი თავი ხდება v_2.
ჩავამატოთ რიგში წვერო (BFS პროცედურის მე-15 სტრიქონი), ის გახდება v_{r+1}, (რადგან Q რიგი შეიცავს $< v_1, v_2, \ldots, v_r >$ წვეროებს), ამ მომენტში, u წვერო, რომლის მოსაზღვრე წვეროთა სია განიხილებოდა, უკვე ამოღებულია რიგიდან და ინდუქციის ჰიპოთეზის თანახმად, რიგის ახალი v_1 თავისთვის, სრულდება უტოლობა: $d[v_1] \leq d[u]$. ამრიგად, $d[v_{r+1}] = d[v] = d[u] + 1 \leq d[v_1] + 1$, გარდა ამისა, ინდუქციის ჰიპოთეზის თანახმად, $d[v_r] \leq d[u] + 1$, და $d[v_r] \leq d[u] + 1 = d[v] = d[v_{r+1}]$, ხოლო დანარჩენი უტოლობები რჩება უცვლელი. ამრიგად ლემა სრულდება რიგში ახალი წვეროს ჩამატების შემდეგაც. □

შედეგი 2.1. ვთქვათ, პროცედურა BFS(G,s) შესრულების პროცესში, Q რიგს ემატება ჯერ v_i და შემდეგ v_j წვეროები, მაშინ v_j წვეროს რიგში ჩამატების მომენტში, სრულდება უტოლობა: $d[v_i] \leq d[v_j]$.

თეორემა 2.1. (სიგანეში ძებნის კორექტულობა) ვთქვათ, მოცემულია G=(V,E) გრაფი (ორიენტირებული ან არაორიენტირებული) და ვთქვათ, სრულდება პროცედურა BFS(G,s) s ფესვის მქონე G=(V,E) გრაფისათვის. მაშინ მუშაობის პროცესში BFS(G,s) ხსნის s-დან მიღწევად ყველა $v \in V$ წვეროს, და პროცედურის დამთავრების შემდეგ, ყველა $v \in V$ წვეროსთვის, შესრულდება ტოლობა $d[v] = \delta(s, v)$. ამის გარდა, s-დან მიღწევადი ნებისმიერი $v \neq s$ თვის ერთ-ერთი უმოკლესი გზა s-დან v-მდე არის უმოკლესი გზა s-დან π[v]-მდე, რომელსაც მოსდევს წიბო $(\pi[v], v)$.

Proof. დავუშვათ საწინააღმდეგო. ვთქვათ, რომელიმე წვეროსთვის, d მნიშვნელობა არ უდრის უმოკლესი გზის სიგრძეს. ვთქვათ, v არის მინიმალური $\delta(s, v)$ სიგრძის მქონე წვერო, იმ წვეროებს შორის, რომლისთვისაც არ არის სწორად გამოთვლილი d მნიშვნელობა. ცხადია, $v \neq s$. **ლემა** 2.2 თანახმად, $d[v] \geq \delta(s, v)$, ამიტომ, ჩვენი დაშვების გამო, $d[v] > \delta(s, v)$. v წვერო მიღწევადი უნდა იყოს s-დან, რადგან წინააღმდეგ შემთხვევაში, $\delta(s, v) = \infty \geq d[v]$. ვთქვათ, u არის წვერო, რომელიც უშუალოდ წინ უძღვის v წვეროს s-დან v-მდე უმოკლეს გზაზე, ასე, რომ შესრულდება $\delta(s, v) = \delta(s, u) + 1$. რადგანაც $\delta(s, u) = \delta(s, v)$, ხოლო v არის მინიმალური $\delta(s, v)$ სიგრძის მქონე წვერო, რომლისთვისაც არ არის სწორად გამოთვლილი d მნიშვნელობა, ამიტომ $d[u] = \delta(s, u)$. საბოლოოდ, მივიღებთ:

$$d[v] > \delta(s, v) = \delta(s, u) + 1 = d[u] + 1 \tag{2.1}$$

ახლა, განვიხილოთ მომენტი, როცა პროცედურა BFS(G,s) იღებს u წვეროს Q რიგიდან მე-9 სტრიქონში. ამ მომენტში v წვერო შეიძლება იყოს თეთრი, რუხი ან შავი. ვაჩვენოთ, რომ სამივე შემთხვევისთვის მივიღებთ წინააღმდეგობას (2.1)-თან.
თუ v წვერო თეთრია, მაშინ მე-13 სტრიქონში სრულდება მინიჭება d[v] = d[u] + 1, რომელიც ეწინააღმდეგება (2.1)-ს. თუ v წვერო შავია, მაშინ ის უკვე ამოღებულია რიგიდან და **შედეგი** 2.1-ს თანახმად, $d[v] \leq d[u]$, რაც, აგრეთვე, ეწინააღმდეგება (2.1)-ს. თუ v წვერო რუხია, მას ეს ფერი შეეძლო მიეღო რიგიდან ω წვეროს ამოშლის დროს. ω წვერო ამოშლილია u წვეროს ამოშლამდე და მისთვის სრულდება ტოლობა $d[v] = d[\omega] + 1$. **შედეგი** 2.1-დან კი გამომდინარეობს, რომ $d[\omega] \leq d[u]$, ამიტომ $d[v] \leq d[u] + 1$, რომელიც ასევე ეწინააღმდეგება (2.1)-ს.
ამრიგად, ყოველი $v \in V$ წვეროსთვის, სრულდება ტოლობა $d[v] = \delta(s, v)$. დამტკიცების დასასრულებლად შევნიშნოთ, რომ თუ π[v]=u, მაშინ d[v]=d[u]+1. ამიტომ, უმოკლესი გზა s-დან v-მდე შეგვიძლია მივიღოთ, თუ ვიპოვით უმოკლეს გზას s-დან π[v]-მდე და შემდეგ გავივლით $(\pi[v], v)$ წიბოზე. □

s წვეროს მქონე G=(V,E) გრაფისათვის, განვიხილოთ G_π **წინამორბედობის ქვეგრაფი** (predecessor subgraph), როგორც $G_\pi = (V_\pi, E_\pi)$, სადაც:

$$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\}$$

$$E_\pi = \{(\pi[v], v) : v \in V_\pi - \{s\}\}$$

G_π ქვეგრაფი წარმოადგენს **სიგანეში ძებნის ხეს** (breadth-first tree), თუ V_π შედგება s-დან მიღწევადი წვეროებისგან და ყოველი $v \in V_\pi$ წვეროსათვის G_π-ში არსებობს ერთადერთი მარტივი გზა s-დან v-ში, რომელიც ერთდროულად არის უმოკლესი გზა s-დან v-ში G გრაფში. სიგანეში ძებნის ხე წარმოადგენს ხეს, რადგან ის ბმულია და $|E_\pi| = |V_\pi| - 1$.
შემდეგი ლემა აჩვენებს, რომ s წვეროს მქონე G=(V,E) გრაფისათვის BFS(G,s) პროცედურის შესრულების შემდეგ, წინამორბედობის ქვეგრაფი წარმოადგენს სიგანეში ძებნის ხეს.

ლემა 2.4. G=(V,E) გრაფისათვის BFS(G,s) პროცედურის შესრულების შემდეგ, პროცედურა π-ს აგებს ისე, რომ წინამორბედობის ქვეგრაფი $G_\pi = (V_\pi, E_\pi)$ წარმოადგენს სიგანეში ძებნის ხეს.

Proof. მე-14 სტრიქონში π[v]=u მინიჭება ხორციელდება მაშინ და მხოლოდ მაშინ, როცა $(u, v) \in E$ და $\delta(s, v) < \infty$, ანუ როცა v მიღწევადია s-დან. ამიტომ, V_π შედგება V-ს იმ წვეროებისგან, რომლებიც მიღწევადია s-დან. რადგან G_π წარმოადგენს ხეს, **თეორემა** 1.1-ს მიხედვით, ის შეიცავს ერთადერთ გზას s-დან V_π სიმრავლის ყოველ წვერომდე. **თეორემა** 2.1-ს გამოყენებით, კი დავასკვნით, რომ ყოველი ასეთი გზა უმოკლესია. □

შემდეგი პროცედურა ბეჭდავს s-დან v-მდე ყველა წვეროს, მას შემდეგ, რაც BFS(G,s) პროცედურით უკვე აგებულია სიგანეში ძებნის ხე.

Algorithm 2: PRINT PATH

Input: გრაფი $G = (V, E)$, წვეროები s და v, მშობლების მასივი π
Output: გზა s-დან v-ში

```
1 PRINT-PATH(v) :
2 |  if v == s :
3 |  |  print(s);
4 |  elif π[v] == NIL :
5 |  |  print(' გზა არ არსებობს');
6 |  else:
7 |  |  PRINT-PATH(π[v]);
8 |  |  print(v);
```

ფუნქციის მუშაობის დრო დასაბეჭდი გზის სიგრძის პროპორციულია, რადგან ყოველი რეკურსიული გამოძახება ერთი ერთეულით ამცირებს გზას ფესვამდე.

2.2 სიღრმეში ძებნის ალგორითმი

სიღრმეში ძებნის ალგორითმი (depth-first search) იწყებს გრაფის წვეროების შემოვლას ნებისმიერი წვეროდან, ყოველ იტერაციაზე ალგორითმი გადადის მიმდინარე წვეროს მოსაზღვრე წვეროზე და ეს პროცესი გრძელდება მანამ, სანამ არ მიაღწევს ჩიხს, ანუ ისეთ მდგომარეობას, როცა წვეროს არ ჰყავს გაუვლელი მოსაზღვრე წვერო. ამ შემთხვევაში, ალგორითმი ბრუნდება ერთი წიბოთი უკან, წვეროში, რომლიდანაც ის მოხვდა ჩიხში და აგრძელებს პროცესს იმ ადგილიდან. ალგორითმი ასრულებს მუშაობას, როცა ბრუნდება საწყის წვეროში, რომელიც ამ მომენტისთვის უკვე ჩიხია. თუ გრაფში დარჩენილია გაუვლელი წვეროები, ალგორითმი ირჩევს ერთ-ერთ მათგანს და პროცესი მეორდება.
სიღრმეში ძებნის სტრატეგია შეიძლება ასეც ჩამოვაყალიბოთ: ვიაროთ გრაფში წინ (სიღრმეში), სანამ არსებობს გაუვლელი წიბო. თუ გაუვლელი წიბო აღარ არსებობს, დავბრუნდეთ უკან და ვეძებოთ სხვა გზა. ასე გავაგრძელოთ მანამ, სანამ არ ამოიწურება ყველა წვერო, რომელიც მიღწევადია საწყისიდან. თუკი გაუვლელი წვერო მაინც დარჩა, ავიღოთ ერთ-ერთი და გავიმეოროთ პროცესი, სანამ არ იქნება შემოვლილი გრაფის ყველა წვერო.
განივად ძებნის მსგავსად, როცა პირველად გამოვლინდება მოსაზღვრე v წვერო, u-ს მნიშვნელობა თავსდება $\pi[v]$-ში. მიიღება ხე (ან ხეები - თუკი ძებნა განმეორდება რამდენიმე წვეროდან). სიღრმეში ძებნის პროცესში მიიღება წინამორბედობის ქვეგრაფი, რომელიც ასე განისაზღვრება:

$$G_\pi = (V, E_\pi) \text{ , სადაც } E_\pi = \{(\pi[v], v) : v \in V \text{ და } \pi[v] \neq NIL\}$$

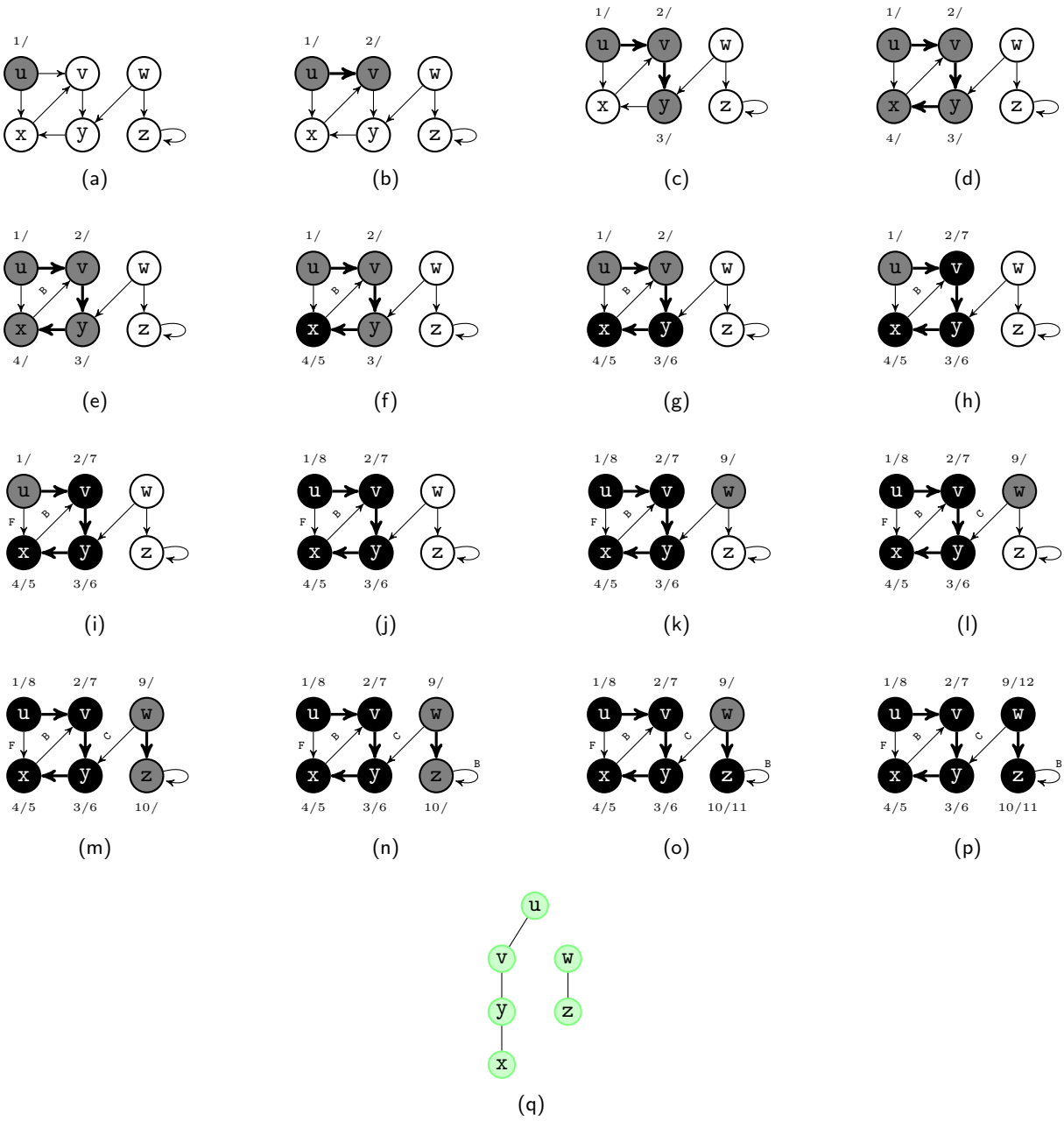
წინამორბედობის ქვეგრაფი წარმოადგენს სიღრმეში ძებნის ტყეს, რომელიც შეიძლება შედგებოდეს სიღრმეში ძებნის ხეებისაგან. E_π სიმრავლეში შემავალ წიბოებს უწოდებენ **ხის წიბოებს**.
სიღრმეში ძებნის ალგორითმიც იყენებს წვეროების შეფერვას. თავიდან ყველა წვერო თეთრია. წვეროს აღმოჩენის შემდეგ იგი ხდება რუხი. როცა წვერო მთლიანად დამუშავებულია, ანუ თუ მისი მოსაზღვრე წვეროების სია ბოლომდე განხილულია, იგი ხდება შავი. ყოველი წვერო ხვდება სიღრმეში ძებნის მხოლოდ ერთ ხეში (ამიტომ ეს ხეები არ გადაიკვეთებიან).
გარდა ამისა სიღრმეში ძებნა თითოეულ წვეროს მიუწერს დროის **ჭდეებს** (timestamp). ყოველ წვეროს აქვს ორი ჭდე: $d[u]$-ში ჩაიწერება, თუ როდის იქნა აღმოჩენილი წვერო (და გახდა რუხი), ხოლო $f[u]$-ში - როდის დასრულდა წვეროს დამუშავება (და გახდა შავი).
შეიძლება დროის ჭდეების ასეთი ინტერპრეტაციაც: მოვათავსოთ წვერო სტეკში, მისი აღმოჩენის მომენტში, და ამოვიღოთ სტეკიდან მაშინ როცა ის ხდება ჩიხი.
ქვემოთ მოყვანილ DFS პროცედურაში $d[u]$ და $f[u]$ წარმოადგენენ მთელ რიცხვებს 1-დან $2|V|$-მდე. ნებისმიერი წვეროსათვის სრულდება უტოლობა $d[u] < f[u]$. u წვერო იქნება თეთრი $d[u]$ მომენტამდე, $d[u]$-სა და $f[u]$-ს შორის იქნება რუხი, ხოლო $f[u]$-ის შემდეგ შავი.
მიმდინარე დროის time ცვლადი გლობალურია და გამოიყენება დროის ჭდეებისათვის.

Algorithm 3: Depth First Search (DFS)

Input: გრაფი $G = (V, E)$
Output: გრაფის სიღრმეში ძებნისას აღმოჩენილი წვეროების მიმდევრობა

```
1  DFS(G) :
2      for ∀u ∈ V :
3          color[u] = TeTri;
4          π[u] = NIL;
5      time = 0;
6      for ∀u ∈ V :
7          if color[u] == TeTri :
8              DFS-VISIT(u)
9      return d, f, π
10 DFS-VISIT(u) :
11     color[u] = ruxi;
12     time++; d[u] = time;
13     for ∀v ∈ Adj[u] :
14         if color[v] == TeTri :
15             π[v] = u;
16             DFS-VISIT(v);
17     color[u] = Savi;
18     time++; f[u] = time;
```

2-4 სტრიქონებში ყველა წვერო ხდება თეთრი, ხოლო π-ს მიენიჭება მნიშვნელობა NIL. მე-5-ე სტრიქონში განისაზღვრება საწყისი (ნულოვანი) დრო. 6-8 სტრიქონებში ხდება პროცედურა DFS-VISIT-ს გამოძახება იმ წვეროებისათვის, რომლებიც თეთრია გამოძახების მომენტისათვის (პროცედურის წინა გამოძახებამ ისინი შეიძლება შავი გახადოს). ეს წვეროები წარმოადგენენ სიღრმეში ძებნის ხეთა ფესვებს.
DFS-VISIT(u) პროცედურის გამოძახებისას u წვერო თეთრია. მე-11 სტრიქონში ის რუხი ხდება. მე-12-ე სტრიქონში მისი აღმოჩენის დრო შეიტანება $d[u]$-ში (წინასწარ დროის მთვლელი 1-ით იზრდება). 13-16 სტრიქონებში განიხილება u-ს მოსაზღვრე წვეროები და ხდება DFS-VISIT პროცედურის გამოძახება u-ს მოსაზღვრე იმ წვეროებისათვის, რომლებიც თეთრი ფერის არიან გამოძახების მომენტში. u-ს ყველა მოსაზღვრე წვეროს განხილვის შემდეგ, იგი ხდება შავი და $f[u]$-ში იწერება ამ მოვლენის ამსახველი დრო (სტრ. 17, 18).
სურ. 2.2-ზე მოცემულია DFS პროცედურის შესრულების პროცესი და სიღრმეში ძებნის ტყე ორიენტირებული გრაფისათვის. მუქად აღნიშნულია **ხის** წიბოები (tree edges). წიბოები, რომლებიც არ შედიან სიღრმეში ძებნის ტყეში, აღნიშნულნი არიან შემდეგნაირად: **უკუწიბოები** - B (back edges), **ჯვარედინი** წიბოები - C (cross edges), **პირდაპირი** წიბოები - F (forward edges).
შევნიშნოთ, რომ სიღრმეში ძებნის შედეგი შეიძლება დამოკიდებული იყოს წვეროების განხილვის თანმიმდევრობაზე, (DFS პროცედურის მე-6 სტრ.), აგრეთვე, მოსაზღვრე წვეროების განხილვის თანმიმდევრობაზე, (DFS-VISIT პროცედურის მე-13 სტრ.). პრაქტიკაში აღნიშნული გარემოება არ ქმნის პრობლემას, რადგან სიღრმეში ძებნის ალგორითმის ნებისმიერი შედეგი შეიძლება ეფექტურად იქნას გამოყენებული და ფაქტობრივად ერთნაირ შედეგებს იძლევა სიღრმეში ძებნაზე დაფუძნებული ალგორითმებისთვის.



ნახ. 2.2:

დავითვალოთ DFS პროცედურის მუშაობის დრო: ციკლები 2-4 და 6-8 სტრიქონებში მოითხოვენ $\Theta(V)$ დროს. პროცედურა DFS-VISIT-ს გამოძახება ხდება მხოლოდ ერთხელ ყოველი წვეროსთვის. DFS-VISIT(u) პროცედურის გამოძახებისას ციკლი 13-16 სტრიქონებში სრულდება $|Adj[v]|$-ჯერ. და რადგან

$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$

ამიტომ DFS-VISIT პროცედურის 13-16 სტრიქონების შესრულების დრო იქნება $\Theta(E)$. DFS პროცედურის მუშაობის დრო კი - $\Theta(V + E)$.
შევნიშნოთ, რომ სიღრმეში ძებნის ხეებისაგან შედგენილი წინამორბედობის ქვეგრაფი ზუსტად შეესაბამება DFS-VISIT პროცედურის რეკურსიული გამოძახებების სტრუქტურას. სახელდობრ, $u = \pi[v]$ მაშინ და მხოლოდ მაშინ, როცა მოხდა DFS-VISIT(v) გამოძახება წვეროს მოსაზღვრე წვეროების განხილვის დროს.
სიღრმეში ძებნის მეორე მნიშვნელოვანი თვისება იმაში მდგომარეობს, რომ წვეროთა აღმოჩენისა და დამუშავების დამთავრების დროები ქმნიან წესიერ ფრჩხილურ სტრუქტურას. აღვნიშნოთ u წვეროს პოვნა მარცხენა ფრჩხილითა და წვეროს სახელით - "(u", ხოლო მისი დამუშავების დამთავრება წვეროს სახელითა და მარჯვენა ფრჩხილით - "u)". მოვლენათა ჩამონათვალი იქნება ფრჩხილებისაგან წესიერად აგებული გამოსახულება. მაგალითად სურ. 2.3-ზე გამოსახული გრაფისათვის:

$$(s(z(y(xx)y)(ww)z)s)(t(vv)(uu)t)$$

(a) (b)

ნახ. 2.3:

ადგილი აქვს მტკიცებულებებს:

თეორემა 2.2. (ფრჩხილური სტრუქტურის შესახებ) სიღრმეში ძებნის დროს ორიენტირებულ ან არაორიენტირებულ $G = (V, E)$ გრაფში ნებისმიერი ორი u და v წვეროსათვის სრულდება მხოლოდ ერთ-ერთი დებულება შემდეგი სამიდან:

- $[d[u], f[u]]$ და $[d[v], f[v]]$ მონაკვეთები არ გადაიკვეთება
- $[d[u], f[u]]$ მონაკვეთი მთლიანად შედის $[d[v], f[v]]$ შიგნით და u წვერო v-ს შთამომავალია სიღრმეში ძებნის ხეში
- $[d[v], f[v]]$ მონაკვეთი მთლიანად შედის $[d[u], f[u]]$ შიგნით და v წვერო u-ს შთამომავალია სიღრმეში ძებნის ხეში

შედეგი 2.2. v წვერო u-ს შთამომავალია სიღრმეში ძებნის ტყეში ორიენტირებულ ან არაორიენტირებულ $G = (V, E)$ გრაფში მაშინ და მხოლოდ მაშინ, როცა $d[u] < d[v] < f[v] < f[u]$.

თეორემა 2.3. (თეთრი გზის შესახებ). v წვერო u-ს შთამომავალია სიღრმეში ძებნის ტყეში (ორიენტირებულ ან არაორიენტირებულ) $G = (V, E)$ გრაფში მაშინ და მხოლოდ მაშინ, როცა $d[u]$ დროის მომენტში (u წვეროს აღმოჩენის), არსებობს გზა u-დან v-ში, რომელიც შედგება მხოლოდ თეთრი წვეროებისაგან.

Proof. დავუშვათ, v წვერო u-ს შთამომავალია. ვთქვათ, w ნებისმიერი წვეროა u-დან v-ში მიმავალ გზაზე სიღრმეში ძებნის ტყეში, ასე რომ w წარმოადგენს u-ს შთამომავალს. **შედეგი** 2.2-ს თანახმად, $d[u] < d[w]$, ამიტომ $d[u]$ მომენტში, w წვერო თეთრია.
ახლა, დავუშვათ, რომ $d[u]$ მომენტში, არსებობს გზა u-დან v-ში, რომელიც შედგება მხოლოდ თეთრი წვეროებისაგან, მაგრამ v წვერო არ ხდება u-ს შთამომავალი სიღრმეში ძებნის ხეში. ზოგადობის შეუზღუდავად შეგვიძლია ვიგულისხმოთ, რომ u-დან v-მდე გზის ყველა დანარჩენი წვერო ხდება u-ს შთამომავალი (წინააღმდეგ შემთხვევაში, v-ს როლში ავიღებთ აღნიშნულ გზაზე u-სთან უახლოეს წვეროს, რომელიც არ ხდება u-ს შთამომავალი). ვთქვათ, w არის v წვეროს წინამორბედი ამ გზაზე, ე.ი. ის u-ს შთამომავალია. (u და w შეიძლება სინამდვილეში ერთი წვერო იყოს). **შედეგი** 2.2-ის თანახმად, $f[w] \leq f[u]$. შევნიშნოთ, რომ v წვეროს აღმოჩენა

ხდება მას შემდეგ, რაც აღმოჩენილია u, მაგრამ w წვეროს დამუშავების დამთავრებამდე. ამრიგად, $d[u] < d[v] < f[w] < f[u]$. **თეორემა** 2.2-ის თანახმად, $[d[v], f[v]]$ მონაკვეთი მთლიანად ეკუთვნის $[d[u], f[u]]$ მონაკვეთს. **შედეგი** 2.2-ის თანახმად კი, v წვერო გახდება u-ს შთამომავალი. □

გრაფის შემოვლის ალგორითმების გამოყენება

ბმული კომპონენტები

როგორც ვიცით, არაა აუცილებელი, რომ გრაფში ყველა ნაწილი ბმული იყოს, ანუ შეიძლება არსებობდეს ორი წვერო, რომელთა შორის შემაერთებელი გზა არ მოიძებნება. ასეთი ცალკეული კომპონენტების პოვნა ფუნდამენტური ამოცანაა გრაფთა თეორიაში: როდესაც რაიმე ამოცანა დაყოფილ გრაფებზე უნდა ამოიხსნას, უმეტეს შემთხვევაში უნდა დავადგინოთ დამოუკიდებელი ნაწილები და ისინი ცალ-ცალკე დავამუშაოთ.
მაგალითად, თუ მოცემულია რაიმე სიმრავლე, მასზე მოცემული ექვივალენტობის მიმართება, როგორც ვიცით, ამ სიმრავლეს ექვივალენტურობის კლასებად ყოფს და თუ ამ მიმართებას გრაფის სახით გამოვხატავთ, თითო კლასი ამ გრაფის ბმულობის კომპონენტი იქნება.

როგორც სიღრმეში, ასევე სიგანეში ძებნის ალგორითმების გამოყენებით ადვილად შეიძლება გრაფის ბმულობის კომპონენტების გამოყოფა: ავირჩევთ ნებისმიერ წვეროს და ამ რომელიმე ალგორითმით შემოვივლით დანარჩენ შესაძლო წვეროებს, რომლებსაც ერთი და იგივე ნიშანს დავადებთ (მაგალითად, მთვლელის რიცხვი). თუ გრაფში სხვა წვეროებიც დარჩა, მთვლელს ერთით გავზრდით და იგივე პროცედურას გავიმეორებთ მანამ, სანამ გრაფის ყველა წვეროს რაიმე ნიშანი არ დაედება.

სავარჯიშო 2.1: დაწერეთ ბმულობის კომპონენტების გამოყოფის ალგორითმი, დაამტკიცეთ მისი სისწორე და გამოითვალეთ ბიჯების ზედა ზღვარი.

ხეებისა და ციკლების პოვნა

ხეები - აციკლური (უციკლო) გრაფები - საინტერესო გრაფთა ყველაზე მარტივ კლასს ქმნიან. სიღრმეში ძებნის მეთოდი პირდაპირ გვაძლევს იმის პასუხს, არის თუ არა მოცემული გრაფი ხე: თუ ძებნის პროცესში განვლილ წიბოს აღვნიშნავთ როგორც „ხის წიბოს", ხოლო ისეთ წიბოს, რომელსაც არ გადავივლით (მაგალითად ისეთს, რომელსაც ერთი წვეროდან უკვე შესწავლილ წვეროში მივყავართ) აღვნიშნავთ, როგორც „დამატებითს", მოცემული გრაფი იქნება ხე მაშინ და მხოლოდ მაშინ, თუ სიღრმეში ძებნის შემდეგ დამატებითი წიბოები არ აქვს. რადგან ნებისმიერი ხისათვის $|E| = |V| - 1$, ამ ალგორითმის დროის ზედა ზღვარი იქნება $O(|V|)$.
თუ გრაფი შეიცავს ციკლს, მისი აღმოჩენა შეიძლება პირველივე დამატებითი წიბოს პოვნით: თუ აღმოჩნდა დამატებითი წიბო (u, v), მაშინ უკვე შექმნილ ხეში უნდა არსებობდეს გზა u წვეროდან v წვეროში და იგი (u, v) წიბოსთან ერთად მოგვცემს ციკლს.

სავარჯიშო 2.2: დაამტკიცეთ, რომ ნებისმიერი დამატებითი (u, v) წიბოს წვერეობს შორის არსებობს შემაერთებელი გზა.

სავარჯიშო 2.3: სიღრმეში ძებნის გამოყენებით დაწერეთ ალგორითმი, რომელიც მოცემული გრაფისათვის დაადგენს, არის თუ არა იგი ხე და თუ არა, ციკლებსაც იპოვნის.

სავარჯიშო 2.4: განიხილეთ წინა ამოცანაში დაწერილი ალგორითმი. შეიძლება თუ არა მისი საშუალებით ყველა ციკლის აღმოჩენა?

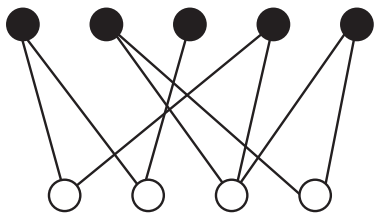
სავარჯიშო 2.5: შეიძლება თუ არა იგივე ამოცანის სიგანეში ძებნის მეთოდით გადაჭრა?

ორად შეღებილი გრაფები

გრაფის შეღებვის ზოგად ამოცანაში გრაფის წვეროები ისე უნდა შეიღებოს, რომ წიბოთი შეღებილ ორ წვეროს სხვადასხვა ფერი ქონდეს. ცხადია, თუ ყველა წვეროს სხვადასხვა ფრად შევღებავთ, ეს ამოცანა გადაიჭრება, მაგრამ საინტერესოა გრაფის რაც შეიძლება ცოტა ფრად შეღებვის ამოცანა - მოკლედ: გრაფის შეღებვის ამოცანა - რომელიც ფართოდ გამოიყენება ალგორითმების თეორიასა თუ პრაქტიკაში:

- ეფექტური ცხრილების შედგენაში - მაგალითად, მრავალბირთვიან პროცესორებში ამოცანის ნაწილების პარალელურად დამუშავების მიზნით - რა ნაწილი რის შემდეგ უნდა დამუშავდეს;
- რადიოტალღების სიხშირეების ეფექტურ დადგენაში - მაგალითად, თუ ორი მომხმარებელი რადიოგადამცემს ხმარობს, ახლოს მდგომებს სხვადასხვა სიხშირეები უნდა ქონდეთ, შორს მდგომებს შეიძლება ერთი და იგივე;
- რეგისტრების ეფექტურად დანაწილების ამოცანა - პროცესორის რეგისტრების გამოყენებით მონაცემების დამუშავება გაცილებით უფრო სწრაფად შეიძლება, ვიდრე RAM მეხსიერებიდან. მაგრამ რადგან ხშირად ცვლადთა რაოდენობა რეგისტრების რაოდენობას აჭარბებს, საჭიროა იმის დადგენა, რა დროს რომელი ცვლადი ჩაიწეროს რეგისტრებში;
- სახეთა ამოცნობაში - მაგალითად, მოცემული სურათით კატალოგში ადამიანის მოძებნა;
- არქეოლოგიური ან ბიოლოგიური მასალის ანალიზი - როგორც ბიოლოგიაში, ასევე არქეოლოგიაში მონაცემები ხის სახით შეგვიძლია შევინახოთ: ერთი სახეობა ან კულტურა მეორედან მომდინარეობს, ერთი სახეობა ან კულტურა სხვა რამოდენიმეს წარმოშობს.

ზოგადად, გრაფის მინიმალურად შეღებვის ამოცანა (ან მისი ***ქრომატული რიცხვის*** დადგენის ამოცანა, როგორც მას უწოდებენ ხოლმე), ძნელი გადასაჭრელია. მისი ერთ-ერთი მნიშვნელოვანი ქვეამოცანაა, შეიძლება თუ არა მოცემული გრაფის ორ ფრად შეღებვა, ან, სხვა სიტყვებით რომ ვთქვათ, ისეთ ორ ნაწილად დაყოფა, რომელშიც წვეროები ერთმანეთისგან იზოლირებულნი არიან. ასეთი გრაფის მაგალითია მოყვანილი ნახაზში 2.4.



ნახ. 2.4: ორად შეღებილი გრაფის მაგალითი

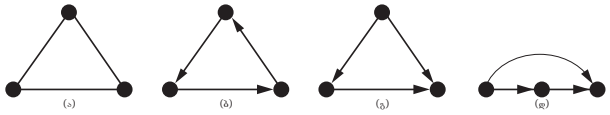
ორად შეღებილ გრაფს ზოგჯერ *ორად დაყოფილსაც* უწოდებენ. იმის დასადგენად, შეიძლება თუ არა მოცემული გრაფის ორად დაყოფა, შემდეგი სტრატეგიით შეიძლება მოქმედება:
ვირჩევთ ერთ-ერთ წვეროს, რომელსაც ვღებავთ რომელიმე ფრად (დავუშვათ, თეთრად). სიღრმეში ან სიგანეში ძებნით ახლად აღმოჩენილ წვეროს ვღებავთ მისი მშობლის (იმ წვეროსი, რომელთანაც არის დაკავშირებული) განსხვავებული ფერით (თეთრი ან შავი). შემდეგ *ყოველი აღმოუჩენელი* წვეროსათვის ვამოწმებთ, არის თუ არა იგი მიერთებული ორ ერთსა და იმავე ფრად შეღებილ წვეროსთან და თუ ასეა, ეს იმას უნდა ნიშნავდეს, რომ გრაფი არ შეიღებება (არ დაიყოფა) ორად. თუ ალგორითმი ამ სახის კონფლიქტის გარეშე დასრულდება, ეს იმას უნდა ნიშნავდეს, რომ გრაფი ორად შეიღება.

სავარჯიშო 2.6: დაამტკიცეთ ამ მეთოდის სისწორე. დამოკიდებულია თუ არა ეს მეთოდი იმაზე, თუ რომელ საწყის წვეროს ავიღებთ?

სავარჯიშო 2.7: ამ მეთოდზე დაყრდნობით დაწერეთ ალგორითმი, რომელიც დაადგენს, შეიძლება თუ არა გრაფის ორად შეღებვა.

ტოპოლოგიური დალაგება

განვიხილოთ ნახ. 2.5-ში მოყვანილი სამი გრაფის მაგალითი.
პირველი გრაფი არაა მიმართული და შეიცავს ციკლს; მეორე მიმართულია და ციკლს შეიცავს, ხოლო მესამე კი მიმართულია, მაგრამ ციკლს არ შეიცავს (აციკლურია), რადგან ვერ მოვძებნით ისეთ *დაშვებულ* გზას, რომელიც გაყვება ისრებს და რომელიმე წვეროდან ისევ იგივე წვეროში დაგვაბრუნებდა.



ნახ. 2.5: არამიმართული ციკლური, მიმართული ციკლური და მიმართული აციკლური გრაფები

ასეთ სტრუქტურას *აციკლური მიმართული გრაფი* ეწოდება და მათი დახაზვა შეიძლება ისე, რომ ყველა წიბო *მარცხნიდან მარჯვნივ* იყოს მიმართული (მაგალითად, ნახ. 2.5(დ)), რასაც ამ გრაფის ***ტოპოლოგიურ დალაგებას*** უწოდებენ.

აღსანიშნავია, რომ მხოლოდ აციკლურ მიმართულ გრაფებს შეიძლება მოვუძებნოთ ტოპოლოგიური დალაგება, რადგან ნებისმიერი ციკლი რომელიმე კვანძიდან *უკან* (ანუ მარჯვნიდან მარცხნივ) დაგვაბრუნებდა, თანაც აციკლურ მიმართულ გრაფებს *ყოველთვის* მოვუძებნით ერთ ტოპოლოგიურ დალაგებას მაინც.

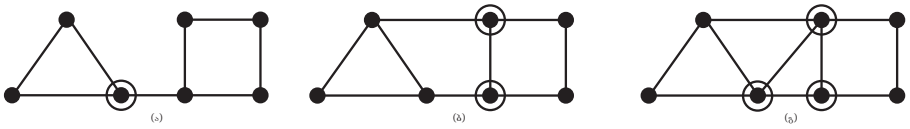
ტოპოლოგიურ დალაგებას დიდი მნიშვნელობა აქვს მთელ რიგ პრაქტიკულ ამოცანებში, მაგალითად ზემოთ ნახსენებ ცხრილის შედგენაში.

შიღრმეში ძებნის ალგორითმით ადვილად შეგვიძლია მიმართული გრაფის აციკლურობის დადგენა (რაც დამოკიდებულია იმაზე, შეგვხვდება თუ არა მუშაობის პროცესში ზემოთ ნახსენები „დამატებითი წიბოები"). თუ გრაფი აციკლურია, მაშინ ალგორითმის მსვლელობისას შექმნილი R მიმდევრობა ტოპოლოგიური დალაგების თანმიმდევრობას გვაძლევს.

სავარჯიშო 2.8: დაამტკიცეთ, რომ სიღრმეში ძებნის პროცესში შექმნილი R მიმდევრობა ტოპოლოგიური დალაგების თანმიმდევრობას გვაძლევს.

საარტიკულაციო კვანძები

ხშირად საჭიროა იმის დადგენა, თუ მინიმუმ რამდენი კვანზის ამოგდებაა საჭირო ბმული გრაფიდან იმისათვის, რომ იგი ნაწილებად დაიშალოს. ქვემოთ მოყვანილ ნახაზში ნაჩვენებია სამი გრაფი, რომელთა დაშლა *მინიმუმ* ერთი (ა), ორი (ბ) და სამი (გ) კვანძის ამოგდებით შეიძლება. ამ შემთხვევაში იტყვიან, რომ გრაფია *ერთად* ბმული, ან *ორად* ბმული, *სამად* ბმული და, ზოგადად, n-ად ბმული.
იტყვიან ასევე, რომ გრაფის ბმულობის კოეფიციენტია n.



ნახ. 2.6: ერთად, ორად და სამად ბმული გრაფი

თუ გრაფის ბმულობის კოეფიციენტია 1, მაშინ იტყვიან, რომ მას აქვს ე.წ. ***საარტიკულაციო კვანძი***.

საარტიკულაციო კვანძების ძებნა ძალიან მნიშვნელოვანია მაგალითად საკომუნიკაციო ქსელების სტაბილურობის დადგენაში. ზოგადად, რაც უფრო მაღალია ქსელის შესაბამისი გრაფის ბმულობის კოეფიციენტი, მით უფრო სტაბილურია იგი.
ადვილი შესამჩნევია, რომ საარტიკულაციო კვანძების ძებნა გრაფიდან რიგ-რიგობით წვეროების ამოგდებითა და დარჩენილი სტრუქტურის ბმულობაზე შემოწმებით შეიძლება.

სავარჯიშო 2.9: დაწერეთ ალგორითმი, რომლითაც გრაფის საარტიკულაციო კვანძების არსებობას დავადგენთ. დაამტკიცეთ მისი სისწორე და დაითვალეთ ბიჯების ზედა ზღვარი.

2.3 წიბოთა კლასიფიკაცია

გრაფის წიბოები იყოფა რამდენიმე კატეგორიად იმის მიხედვით, თუ რა როლს თამაშობენ ისინი სიღრმეში ძებნის დროს. ეს კლასიფიკაცია სასარგებლოა სხვადასხვა ამოცანების განხილვისას. მაგალითად, ორიენტირებულ გრაფს არ გააჩნია ციკლები მაშინ და მხოლოდ მაშინ, როცა სიღრმეში ძებნის ალგორითმი ვერ პოულობს მასში "უკუწიბოებს".
G გრაფისთვის G_π სიღრმეში ძებნის ტყის გამოყენებით შეიძლება განვსაზღვროთ წიბოების ოთხი ტიპი:

1. **ხის წიბოები** (tree edges) - ესაა G_π გრაფის წიბოები. (u, v) წიბო იქნება ხის წიბო, თუ v წვერო აღმოჩენილია ამ წიბოს დამუშავების დროს.

2. **უკუწიბოები** (back edges) - ესაა (u, v) წიბოები, რომლებიც აერთებენ u წვეროს მის v წინაპართან სიღრმეში ძებნის ხეზე. ორიენტირებული გრაფებისათვის დამახასიათებელი მარყუჟები უკუწიბოებად ითვლებიან.

3. **პირდაპირი წიბოები** (forward edges) - ესაა (u, v) წიბოები, რომლებიც აერთებენ წვეროს მის შთამომავალთან, მაგრამ არ შედიან სიღრმეში ძებნის ხეში.

4. **ჯვარედინი წიბოები** (cross edges) - გრაფის ყველა სხვა წიბო. მათ შეუძლიათ შეაერთონ სიღრმეში ძებნის ხის ორი ისეთი წვერო, რომელთა შორის არც ერთი არაა მეორის წინაპარი ან წვეროები, რომლებიც სხვადასხვა ხეებს ეკუთვნიან.

2. (ბ) ნახაზზე გრაფი მოცემულია იმდაგვარად, რომ ხის წიბოები და პირდაპირი წიბოები მიემართებიან ქვემოთ, ხოლო უკუწიბოები - ზემოთ.
DFS ალგორითმით შესაძლებელია წიბოთა კლასიფიკაცია. (u, v) წიბოს ტიპი შეიძლება განისაზღვროს v წვეროს ფერით, პირველი დამუშავების დროს (არ ხერხდება მხოლოდ პირდაპირ და ჯვარედინ წიბოებს შორის განსხვავების პოვნა):

1. თეთრი ფერი - ხის წიბო
2. რუხი ფერი - უკუწიბო
3. შავი - პირდაპირი ან ჯვარედინი წიბო

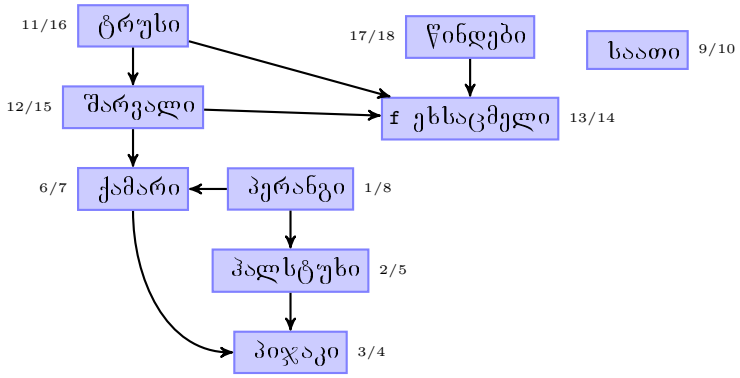
პირველი შემთხვევა უშუალოდ ალგორითმის მუშაობიდან გამომდინარეობს. მეორე შემთხვევისთვის შევნიშნოთ, რომ რუხი წვეროები ყოველთვის ქმნიან შთამომავლების მიმდევრობას, რომლებიც შეესაბამებიან DFS-VISIT პროცედურის გამოძახებებს. რუხი წვეროების რაოდენობა ერთით მეტია სიღრმეში ძებნის ხეზე უკანასკნელად გახსნილი წვეროს სიღრმეზე. წვეროების აღმოჩენა ხდება ამ მიმდევრობის პირველი, "ყველაზე ღრმა", რუხი წვეროდან, ასე რომ წიბო, რომელიც აღწევს სხვა რუხ წვეროს, აღწევს საწყისი წვეროს წინაპარს. მესამე შემთხვევისთვის, პირდაპირი და ჯვარედინი წიბოების განსასხვავებლად შეგვიძლია გამოვიყენოთ d-ს მნიშვნელობა: თუ $d[u] < d[v]$, მაშინ (u, v) წიბო პირდაპირია, ხოლო თუ $d[u] > d[v]$ - ჯვარედინი.
არაორიენტირებული გრაფი საჭიროებს განსაკუთრებულ განხილვას, რადგან ერთი და იგივე წიბო $(u, v) = (v, u)$ ორჯერ უნდა დამუშავდეს (თითოჯერ ორივე ბოლოდან) და შესაძლოა სხვადასხვა კატეგორიაში მოხვდეს. ამ შემთხვევაში ის უნდა მივაკუთვნოთ იმ კატეგორიას, რომელიც ჩვენს ჩამონათვალში უფრო წინ დგას. იმავე შედეგს მივიღებთ, თუკი ჩავთვლით, რომ წიბოს ტიპი განისაზღვრება მისი პირველი დამუშავებისას და არ იცვლება მეორე დამუშავებით. ირკვევა, რომ ასეთი შეთანხმებისას პირდაპირი და ჯვარედინი წიბოები არაორიენტირებულ გრაფში არ იქნება და ადგილი აქვს თეორემას.

თეორემა 2.4. სიღრმეში ძებნისას არაორიენტირებულ G გრაფში ნებისმიერი წიბო ან ხის წიბოა, ან უკუწიბო.

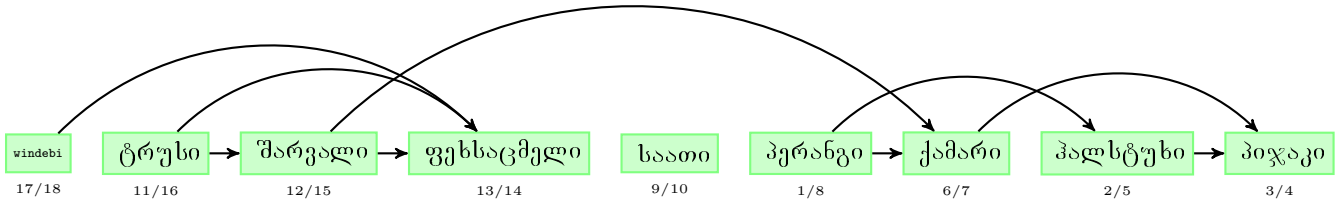
2.4 ტოპოლოგიური სორტირება

ვთქვათ, მოცემულია ორიენტირებული აციკლური გრაფი (directed acyclic graph). ამ გრაფის **ტოპოლოგიური სორტირება** (topological sort) გულისხმობს წვეროების განლაგებას ისეთი წრფივი თანმიმდევრობით, რომ ნებისმიერი წიბო მიმართული იყოს ამ თანმიმდევრობაში ნაკლები ნომრის მქონე წვეროდან მეტი ნომრის მქონე წვეროსაკენ. ცხადია, რომ თუკი გრაფი შეიცავს ციკლს, ასეთი თანმიმდევრობა არ იარსებებს. ამოცანა შეგვიძლია სხვაგვარადაც დავსვათ: განვალაგოთ გრაფის წვეროები ჰორიზონტალურ წრფეზე ისე, რომ ყველა წიბო მიმართული იყოს მარცხნიდან მარჯვნივ.
მაგალითისათვის განვიხილოთ ასეთი შემთხვევა: დაბნეული პროფესორისათვის დილემადაა ქცეული დილაობით ჩაცმა. მან ზოგიერთი რამის ჩაცმისას აუცილებლად უნდა დაიცვას მოქმედებების თანმიმდევრობა (მაგ.: ჯერ

წინდები, შემდეგ ფეხსაცმელი), ზოგ შემთხვევაში კი თანმიმდევრობას მნიშვნელობა არა აქვს (მაგ.: წინდები და შარვალი). სურ. 2.7ა-ზე ეს დამოკიდებულებანი მოცემულია ორიენტირებული გრაფის სახით. (u, v) წიბო აღნიშნავს, რომ u უნდა იქნას ჩაცმული v-ზე ადრე. ამ გრაფის ტოპოლოგიური სორტირების ერთ-ერთი ვარიანტი მოცემულია სურ. 2.7ბ-ზე.



(a)



(b)

ნახ. 2.7:

წვეროების გვერდით მითითებულია მათი დამუშავების დაწყებისა და დამთავრების დროები. სურ. 2.7ბ-ში გრაფი ტოპოლოგიურად სორტირებულია. წვეროები დალაგებულია დამუშავების დამთავრების დროთა მიხედვით. ყველა წიბო მიმართულია მარცხნიდან მარჯვნივ.
შემდეგი ალგორითმი ტოპოლოგიურად ალაგებს ორიენტირებულ აციკლურ გრაფს.

Algorithm 4: Topological Sort

Input: (DAG) ორიენტირებული აციკლური გრაფი $G = (V, E)$
Output: ტოპოლოგიურად სორტირებული წვეროების მიმდევრობა

```
1 თIჩ"-შდთ() :
2 |   L = [];    // ცარიელი სია
3 |   DFS(G);    // წვეროს დამუშავების დამთავრებისას
4 |              //(DFS-VISIT,  სტრ.18)
5 |              // დავამატოთ წვერო  სიის დასაწყისში
6 |   return L
```

ტოპოლოგიური სორტირება სრულდება $\Theta(V+E)$ დროში, რადგან ამდენი დრო სჭირდება სიღრმეში ძებნას, ხოლო სიაში წვეროს ჩაწერას სჭირდება $O(E)$ დრო. ალგორითმის სისწორე მტკიცდება შემდეგი ლემის დახმარებით:

ლემა 2.5. ორიენტირებული გრაფი არ შეიცავს ციკლებს მაშინ და მხოლოდ მაშინ, როცა სიღრმეში ძებნის ალგორითმი ვერ პოულობს მასში უკუწიბოებს.

Proof. ვთქვათ, არსებობს (u, v) უკუწიბო, მაშინ v არის u-ს წინაპარი სიღრმეში ძებნის ხეზე, ამრიგად, გრაფში არსებობს გზა v-დან u-ში და (u, v) წიბო ასრულებს ციკლს.
ვთქვათ, გრაფში გვაქვს ციკლი c. დავამტკიცოთ, რომ ამ შემთხვევაში სიღრმეში ძებნა აუცილებლად იპოვის უკუწიბოს. ციკლის წვეროებიდან ამოვირჩიოთ წვერო v, რომელიც პირველად იქნა აღმოჩენილი, და ვთქვათ, (u, v) c-ში შემავალი წიბოა. $d[v]$ მომენტში v-დან u-ში მივყავართ თეთრი წვეროებისაგან შემდგარ გზას. **თეორემა 2.3 (თეთრი გზის შესახებ)**-ის თანახმად, u გახდება სიღრმეში ძებნის ტყეში, ამიტომ (u, v) იქნება უკუწიბო. □

თეორემა 2.5. *TOPOLOGICAL-SORT*(G) პროცედურა სწორად ასრულებს ტოპოლოგიურ სორტირებას აციკლური ორიენტირებული G გრაფისთვის.

Proof. ვთქვათ, $G = (V, E)$ აციკლური ორიენტირებული გრაფისთვის შესრულდა DFS პროცედურა, რომელიც გამოითვლის მისი წვეროებისთვის დამთავრების დროს. საკმარისია ვაჩვენოთ, რომ თუ ნებისმიერი ორი განსხვავებული u და v წვეროსთვის არსებობს (u, v) წიბო, მაშინ $f[v] < f[u]$. (u, v) წიბოს დამუშავების მომენტში, წვერო v არ შეიძლება იყოს რუხი (ამ შემთხვევაში, ის იქნებოდა u-ს წინაპარი, ხოლო (u, v) წიბო იქნებოდა უკუწიბო, რაც ეწინააღმდეგება **ლემა** 2.5-ს, ამიტომ, (u, v) წიბოს დამუშავების მომენტში, წვერო v უნდა იყოს ან თეთრი, ან შავი. თუ v თეთრია, ის გახდება u-ს შთამომავალი და $f[v] < f[u]$. თუ ის უკვე შავია, მაშინ, $f[v]$-ს მნიშვნელობა უკვე დადგენილია, ხოლო u ჯერ დამუშავების პროცესშია, ამიტომ $f[v] < f[u]$. ამრიგად, აციკლური ორიენტირებული გრაფის ნებისმიერი (u, v) წიბოსთვის სრულდება $f[v] < f[u]$. □

2.5 ძლიერად ბმული კომპონენტები

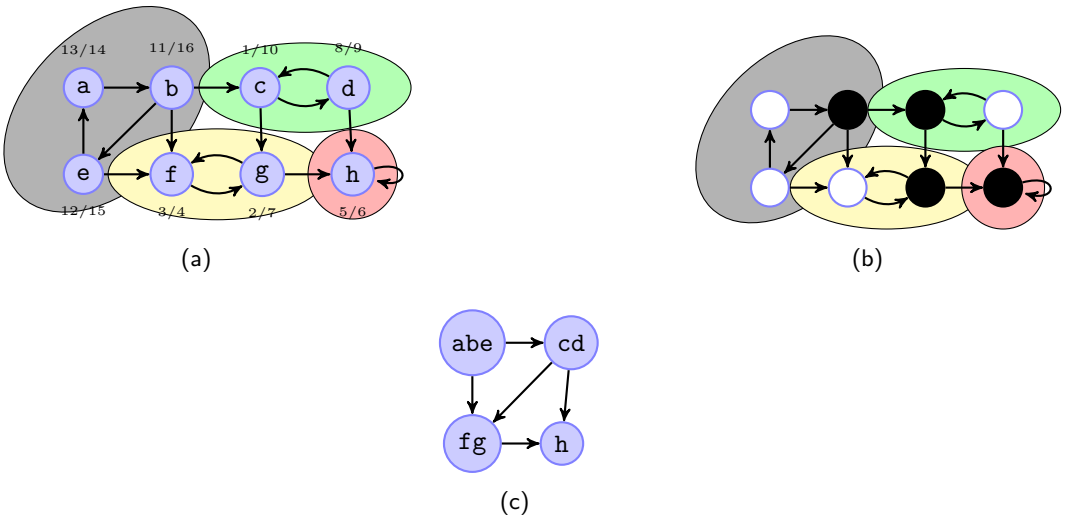
სიღრმეში ძებნის ალგორითმის გამოყენების კლასიკური მაგალითია გრაფის ძლიერად ბმულ კომპონენტებად დაშლა. ორიენტირებულ გრაფებზე მომუშავე მრავალი ალგორითმი იწყება გრაფში ძლიერად ბმული კომპონენტების მოძებნით. ამის შემდეგ ამოცანა იხსნება ცალკეული კომპონენტებისათვის, ხოლო შემდეგ ხდება კომბინირება ამ კომპონენტთა კავშირების შესაბამისად.
გავიხსენოთ, რომ $G = (V, E)$ ორიენტირებული გრაფის ძლიერად ბმული კომპონენტი ეწოდება $U \subset V$ წვეროთა მაქსიმალურ სიმრავლეს, სადაც ნებისმიერი ორი u და v წვერო $(u, v \in U)$ ერთმანეთისაგან მიღწევადია. $G = (V, E)$ გრაფის ძლიერად ბმული კომპონენტების ძებნის ალგორითმი იყენებს "ტრანსპონირებულ" $G^T = (V, E^T)$ გრაფს, რომელიც მიიღება საწყისი გრაფიდან წიბოთა მიმართულებების შებრუნებით. ასეთი გრაფი შეიძლება აიგოს $O(V + E)$ დროში (ვგულისხმობთ, რომ ორივე გრაფი მოიცემა მოსაზღვრე წვეროთა სიით). ცხადია, რომ G და G^T გრაფებს ერთი და იგივე ძლიერად ბმული კომპონენტები აქვთ. შემდეგი ალგორითმი პოულობს ძლიერად ბმულ კომპონენტებს $G = (V, E)$ ორიენტირებულ გრაფში სიღრმეში ძებნის ორჯერ გამოყენებით - G და G^T გრაფებისათვის. ალგორითმის მუშაობის დროა $O(V + E)$.

Algorithm 5: Strongly Connected Components

Input: ორიენტირებული გრაფი $G = (V, E)$
Output: ძლიერად ბმული კომპონენტების წვეროების სია

```
1 STRONGLY-CONNECTED-COMPONENTS(G) :
2     DFS(G);        // ვიპოვოთ f მასივი
3                    // წვეროების დამუშავების დამთავრების დროები
4     ავაგოთ G^T; // ავაგოთ გრაფის ტრანსპონირებული გრაფი
5     DFS(G^T);    // გარე ციკლში DFS str.6
6                  // წვეროები f კლებადობის მიხედვით ჩავიაროთ
7                  // აგებული ძებნის ხეები იქნება
8                  // G გრაფის ძლიერად ბმული კომპონენტები
9     return ძლიერად ბმული კომპონენტების სია
```

სურ. 2.8ა-ზე რუხი ფონით აღნიშნულია G გრაფის ძლიერად ბმული კომპონენტები, ნაჩვენებია აგრეთვე სიღრმეში ძებნის ტყე და დროის ჭდეები G^T გრაფისათვის. სურ. 2.8ბ-ზე მოცემულია G გრაფის სიღრმეში ძებნის ხე, რომელიც პროცედურის მე-5 სტრიქონში გამოითვლება. b, c, g, h წვეროები წარმოადგენენ სიღრმეში ძებნის ხეთა ფესვებს G^T გრაფისათვის და შეფერილი არიან შავად სურ. 2.8ც-ზე მოცემულია აციკლური ორიენტირებული გრაფი, რომელიც მიიღება G გრაფის ძლიერად ბმული კომპონენტების წერტილებამდე შეკუმშვით. მას **კომპონენტების გრაფს** უწოდებენ.
ამ ალგორითმის იდეა ეყრდნობა **კომპონენტების გრაფის** $G^{SCC} = (V^{SCC}, E^{SCC})$ თვისებას, რომელიც შემდეგში მდგომარეობს: ვთქვათ, G გრაფს აქვს ძლიერად ბმული კომპონენტები $C_1, \ldots, C_k$. წვეროთა $V^{SCC} = \{v_1, \ldots, v_k\}$ სიმრავლე შედგება v_i წვეროებისგან გრაფის ყოველი ძლიერად ბმული C_i კომპონენტისთვის. თუ G-ში არსებობს წიბო (x, y) რაიმე ორი წვეროსთვის $x \in C_i$, $y \in C_j$, მაშინ კომპონენტების გრაფში არსებობს წიბო $(v_i, v_j) \in E^{SCC}$. სხვა სიტყვებით რომ ვთქვათ, თუ G გრაფის ყოველ ძლიერად ბმულ კომპონენტში შევკუმშავთ მოსაზღვრე წვეროების დამაკავშირებელ წიბოებს, მივიღებთ G^{SCC} გრაფს, რომლის წვეროებს წარმოადგენენ G გრაფის ძლიერად ბმული კომპონენტები. იხ. სურ. 2.8ც.
კომპონენტების გრაფის ძირითადი თვისება მდგომარეობს იმაში, რომ ეს გრაფი წარმოადგენს აციკლურ ორიენტირებულ გრაფს, რაც გამომდინარეობს შემდეგი ლემიდან:



ნახ. 2.8:

ლემა 2.6. ვთქვათ, C და C' ორიენტირებული G გრაფის განსხვავებული ძლიერად ბმული კომპონენტებია, და ვთქვათ, $u, v \in C$ და $u', v' \in C'$, გარდა ამისა, ვთქვათ, G გრაფში არსებობს გზა u-დან u'-ში. მაშინ G გრაფში არ შეიძლება არსებობდეს გზა v'-დან v-ში.

Proof. თუ G გრაფში არსებობს გზა v'-დან v-ში, მაშინ G-ში არსებობს გზები u-დან v'-ში და v'-დან u-ში. ე.ი. u და v' ერთმანეთისთვის მიღწევადი წვეროებია, რაც ეწინააღმდეგება პირობას, რომ C და C' G გრაფის განსხვავებული ძლიერად ბმული კომპონენტებია. □

რადგან, პროცედურა STRONGLY-CONNECTED-COMPONENTS ორჯერ ასრულებს სიღრმეში ძებნას, ამ თავში ჩავთვალოთ, რომ $d[u]$ და $f[u]$ აღნიშნავენ აღმოჩენის და დამთავრების დროებს DFS პროცედურის პირველი გამოძახების შესრულების დროს (სტრ. 2).
შემოვიღოთ შემდეგი აღნიშვნები: ვთქვათ, $U \subseteq V$, $d(U) = \min_{u \in U}\{d[U]\}$, $f(U) = \max_{u \in U}\{f[u]\}$. ე.ი. $d(U)$ წარმოადგენს წვეროს აღმოჩენის ყველაზე ადრინდელ დროს U-ს წვეროებს შორის, ხოლო $f(U)$ - დამთავრების ყველაზე გვიან დროს U-ს წვეროებს შორის.

ლემა 2.7. ვთქვათ, C და C' ორიენტირებული $G = (V, E)$ გრაფის განსხვავებული ძლიერად ბმული კომპონენტებია, და ვთქვათ, არსებობს წიბო $(u, v) \in E$, სადაც $u \in C$ და $v \in C'$, მაშინ $f(C) > f(C')$.

Proof. იმის მიხედვით, სიღრმეში ძებნის პროცესში ძლიერად ბმულ რომელ კომპონენტში, C-ში თუ C'-შია პირველად აღმოჩენილი წვერო, გვაქვს ორი შემთხვევა:

- თუ $d(C) < d(C')$, აღვნიშნოთ C-ში აღმოჩენილი პირველი წვერო x-ით. $d[x]$ მომენტში ყველა წვერო C-ში და C'-ში თეთრია. G-ში არსებობს გზა x-დან C-ს ყველა წვერომდე, რომელიც შედგება მხოლოდ თეთრი წვეროებისგან. რადგან, $(u, v) \in E$, $d[x]$ მომენტში, ნებისმიერი $\omega \in C'$ წვეროსთვის, G-ში არსებობს აგრეთვე გზა x-დან ω-ში, რომელიც შედგება მხოლოდ თეთრი წვეროებისგან. თეთრი გზის შესახებ თეორემის თანახმად, ყველა წვერო C-ში და C'-ში, ხდება x-ს შთამომავალი სიღრმეში ძებნის ხეზე. **შედეგი** 2.2-ის თანახმად, $f[x] = f(C) > f(C')$.

- თუ $d(C) > d(C')$, აღვნიშნოთ C'-ში აღმოჩენილი პირველი წვერო y-ით. $d[y]$ მომენტში ყველა წვერო C'-ში თეთრია. G-ში არსებობს გზა y-დან C'-ს ყველა წვერომდე, რომელიც შედგება მხოლოდ თეთრი წვეროებისგან. თეთრი გზის შესახებ თეორემის თანახმად, ყველა წვერო C'-ში ხდება y-ს შთამომავალი სიღრმეში ძებნის ხეზე. **შედეგი** 2.2-ის თანახმად, $f[y] = f(C')$. $d[y]$ მომენტში ყველა წვერო C-ში თეთრია. რადგან არსებობს წიბო (u, v) C-დან C'-ში, **ლემა** 2.6-ის თანახმად, არ არსებობს გზა C'-დან C-ში. ე.ი. C-ში არ არის y-დან მიღწევადი წვეროები. ამგვარად, $f[y]$ მომენტში ყველა წვერო C-ში რჩება თეთრი, რაც ნიშნავს, რომ ნებისმიერი $\omega \in C$ წვეროსთვის, გვაქვს $f[\omega] > f[y]$, საიდანაც გამომდინარეობს, რომ $f[x] = f(C) > f(C')$.

□

შედეგი 2.3. ვთქვათ, C და C' ორიენტირებული $G = (V, E)$ გრაფის განსხვავებული ძლიერად ბმული კომპონენტებია, და ვთქვათ, არსებობს წიბო $(u, v) \in E^T$, სადაც $u \in C$ და $v \in C'$ მაშინ $f(C) < f(C')$.

Proof. რადგან $(u,v) \in E^T$ $(v,u) \in E$, G-ს და G^T-ს აქვს ერთი და იგივე ძლიერად ბმული კომპონენტები, ლემა 2.7-დან გამომდინარეობს, რომ $f(C) < f(C')$.
შედეგი 2.3-ს საშუალებით, განვიხილოთ როგორ მუშაობს პროცედურა STRONGLY-CONNECTED-COMPONENTS. როდესაც ვახორციელებთ სიღრმეში ძებნას G^T გრაფზე, ვიწყებთ იმ x წვეროდან, რომლისთვისაც $f[x]$ მაქსიმალურია. ეს წვერო ეკუთვნის რაიმე ძლიერად ბმულ C კომპონენტს. ამასთან, მოხდება C-ს ყველა წვეროს განხილვა. **შედეგი** 2.3-ის თანახმად, G^T გრაფში არ არის წიბო, რომელიც აკავშირებს C-ს სხვა ძლიერად ბმულ კომპონენტთან (რადგან მაშინ $f(C) < f(C')$, რაც ეწინააღმდეგება იმას, რომ $f[x]$ მაქსიმალურია.) ამიტომ x-დან სიღრმეში ძებნის დროს არ ხდება სხვა კომპონენტების წვეროების განხილვა. ამრიგად ხე, რომლის ფესვი არის x, შეიცავს მხოლოდ C-ს წვეროებს. მას შემდეგ, რაც განხილული იქნება C-ს ყველა წვერო, პროცედურის მე-5 სტრიქონში ხდება წვეროს არჩევა იმ სხვა ძლიერად ბმულ C' კომპონენტიდან, რომლისთვისაც $f(C')$ მაქსიმალურია ყველა სხვა კომპონენტთან შედარებით. შედეგი 2.3-ს თანახმად, G^T გრაფში ერთადერთი წიბო, რომელიც დააკავშირებდა C'-ს სხვა კომპონენტებთან შეიძლება იყოს მიმართული C-ში, მაგრამ C-ს დამუშავება უკვე დასრულებულია. ამრიგად, ყოველი სიღრმეში ძებნა ახორციელებს მხოლოდ ერთი ძლიერად ბმული კომპონენტის დამუშავებას. □

თეორემა 2.6. პროცედურა STRONGLY-CONNECTED-COMPONENTS(G) კორექტულად ახორციელებს ძლიერად ბმულ კომპონენტების ძებნას G გრაფში.

Proof. ვისარგებლოთ ინდუქციით სიღრმეში ძებნის დროს G^T გრაფში ხეების რაოდენობის მიხედვით და დავამტკიცოთ, რომ ყოველი ხის ფესვი ქმნის ძლიერად ბმულ კომპონენტს. $k = 0$-თვის ეს მტკიცება სამართლიანია. დავუშვათ, სიღრმეში ძებნის პირველი k ხე (სტრ. 5) წარმოადგენს ძლიერად ბმულ კომპონენტს და განვიხილოთ $k+1$-ე ხე. ვთქვათ, ამ ხის ფესვია u და ვთქვათ, u ეკუთვნის C ძლიერად ბმულ კომპონენტს. რადგან G^T გრაფში სიღრმეში ძებნა ხორციელდება წვეროების $f[u]$ სიდიდის კლებადობის მიხედვით, ამიტომ ნებისმიერი C' ძლიერად ბმული კომპონენტისთვის, რომლის წვეროები ჯერ განხილული არ არის და რომელიც განსხვავებულია C-გან, სამართლიანია: $f[u] = f(C) > f(C')$. ინდუქციის დაშვების თანახმად, u წვეროს აღმოჩენის მომენტში, C-ს ყველა წვერო თეთრია, თეთრი გზის შესახებ თეორემის თანახმად, C-ს ყველა წვერო u-ს გარდა, წარმოადგენს u-ს შთამომავალს სიღრმეში ძებნის ხეზე. გარდა ამისა, G^T-ს ყველა წიბო, რომელიც გამოდის C-დან, მიმართული შეიძლება იყოს უკვე განხილული ძლიერად ბმული კომპონენტების წვეროებისკენ. ამიტომ, არც ერთ C-გან განსხვავებულ ძლიერად ბმულ კომპონენტში არ არის წვერო, რომელიც იქნებოდა u-ს შთამომავალი სიღრმეში ძებნის ხეზე. ამრიგად, G^T-ს u ფესვის მქონე სიღრმეში ძებნის ხის წვეროები, ქმნიან ზუსტად ერთ ძლიერად ბმულ კომპონენტს, რაც ამტკიცებს თეორემას. □

რიგი (Queue) არის დინამიკური სიმრავლე, რომელშიც ელემენტების ჩამატება და წაშლა ნებისმიერ პოზიციაში კი არ ხდება, არამედ განისაზღვრება სიმრავლის სტრუქტურით. რიგიდან შეიძლება მხოლოდ იმ ელემენტის წაშლა, რომელიც მასში პირველი იქნა ჩამატებული, ანუ რიგში ყველაზე დიდხანს იმყოფება, ხოლო ელემენტის ჩამატება ხდება რიგის ბოლოს: რიგი ორგანიზებულია პრინციპით: პირველი მოვიდა - ბოლო წავიდა ანუ FIFO (first-in, first-out).
Q რიგში v ელემენტის დამატების ოპერაცია აღინიშნება როგორც ENQUEUE(Q,v), ხოლო პირველი ელემენტის ამოშლა - DEQUEUE(Q), ამასთან ეს ოპერაცია აბრუნებს პირველი ელემენტის მნიშვნელობას. განისაზღვრება რიგის თავი (head) და ბოლო (tail). ყოველი ახალდამატებული ელემენტი აღმოჩნდება რიგის ბოლოში, ხოლო წასაშლელი თავში. head(Q) არის რიგის დასაწყისის ინდექსი, ხოლო $tail(Q)$ თავისუფალი უჯრის ინდექსი, რომელიც განკუთვნილია ახალი ელემენტის ჩასამატებლად. რიგი შედგება მასივის ელემენტებისგან, რომლებიც დგანან შესაბამისად $head(Q), head(Q)+1, \ldots, tail(Q)-1$ ადგილებზე.
სტეკი (stack) არის დინამიკური სიმრავლე, რომელშიც შეიძლება მხოლოდ ბოლო ელემენტის ამოშლა, ხოლო ელემენტის ჩამატება შეიძლება მხოლოდ მის ბოლოს. ორგანიზებულია პრინციპით: ბოლო მოვიდა - პირველი წავიდა ანუ LIFO (last-in, first-out).

2.6 სავარჯიშოები

1. სურ. 2.9 გრაფებისთვის განახორციელეთ სიგანეში ძებნა, ააგეთ სიგანეში ძებნის ხე.
2. სურ. 2.9 გრაფებისთვის განახორციელეთ სიღრმეში ძებნა. ააგეთ სიღრმეში ძებნის ტყე.
3. ვთქვათ, მოცემული გვაქვს $G = (V, E)$ არაორიენტირებული გრაფი. სამართლიანია თუ არა შემდეგი მტკიცებულებები:
   (ა) სიღრმეში ძებნის ყველა ტყე (დაწყებული სხვადასხვა საწყისი წვეროდან) შეიცავს ხეების ერთი და იგივე რაოდენობას.

ნახ. 2.9:

(ბ) სიღრმეში ძებნის ყველა ტყე შეიცავს ხის წიბოების ერთი და იგივე რაოდენობას.

4. ვთქვათ, მოცემული გვაქვს $G = (V, E)$ ორიენტირებული გრაფი. აჩვენეთ, რომ (u, v) წიბო არის ხის წიბო ან პირდაპირი წიბო მაშინ და მხოლოდ მაშინ, როცა $d[u] < d[v] < f[v] < f[u]$.

5. სურ. 2.10ა გრაფში დაალაგეთ წვეროები ტოპოლოგიური სორტირების ალგორითმით.

ნახ. 2.10:

6. $G = (V, E)$ გრაფში, სადაც $V = \{v, s, w, q, t, x, y, z\}$
$E = \{(v, w), (w, s), (s, v), (q, w), (q, s), (q, t), (t, y), (y, q), (t, x), (x, z), (z, x)\}$ განახორციელეთ სიღრმეში ძებნა და მოახდინეთ წიბოთა კლასიფიკაცია.

7. იპოვეთ ძლიერად ბმული კომპონენტები სურ. 2.10ბ გრაფში.

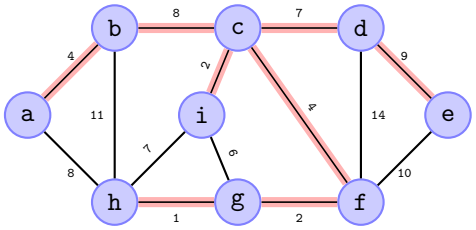
თავი 3

მინიმალური დამფარავი ხეები

ვთქვათ, მოცემულია ბმული არაორიენტირებული $G = (V, E)$ გრაფი. გრაფის ყოველი $(u, v) \in E$ წიბოსათვის მოცემულია $w(u, v)$ არაუარყოფითი წონა. ვიპოვოთ ისეთი ბმული, აციკლური ქვესიმრავლე $T \subseteq E$, რომელიც მოიცავს ყველა წვეროს და რომლისთვისაც ჯამური წონა

$$w(T) = \sum_{(u,v)\in T} w(u, v)$$

მინიმალურია. რადგან T სიმრავლე აციკლურია და აერთებს G გრაფის ყველა წვეროს, ის ქმნის ხეს, რომელსაც უწოდებენ ამ გრაფის **დამფარავ ხეს** (spanning tree). ასეთი T სიმრავლის პოვნის ამოცანას კი უწოდებენ ამოცანას **მინიმალური დამფარავი ხის** (minimum-spanning-tree problem) შესახებ. აქ სიტყვა "მინიმალური" აღნიშნავს "მინიმალურ შესაძლო წონას". შევნიშნოთ, რომ თუკი განვიხილავთ მხოლოდ ხეებს, მაშინ წონათა არაუარყოფითობის პირობა შეგვიძლია უგულებელვყოთ, რადგან ყველა დამფარავ ხეში წიბოთა ერთნაირი რაოდენობაა და შეგვიძლია ერთი და იგივე სიდიდით გავზარდოთ ყველა წიბოს წონა, რაც მათ დადებითად აქცევს.



ნახ. 3.1:

სურ. 3.1-ზე მოცემულია ბმული გრაფისა და მისი მინიმალური დამფარავი ხის მაგალითი, რომლის წიბოებიც გამოყოფილია. მინიმალური დამფარავი ხის ჯამური წონაა 37 და ასეთი ხე ერთადერთი არაა. თუკი (b, c) წიბოს შევცვლით (a, h) წიბოთი, მივიღებთ სხვა დამფარავ ხეს იმავე ჯამური წონით.

ჩვენ განვიხილავთ მინიმალური დამფარავი ხის პოვნის ორ ხერხს: პრიმისა და კრასკალის ალგორითმებს. ორივე ალგორითმი იყენებს "ხარბ" სტრატეგიას - ეძებს "ლოკალურად საუკეთესო" ვარიანტს მუშაობის ყოველ ბიჯზე.

ჩვენი ალგორითმების ზოგადი სქემა ასეთია: ვაშენებთ დამფარავი ხე აიგება A სიმრავლეს ყოველ ბიჯზე თანდათანობით - თავდაპირველად A ცარიელ ემატება თითო წიბო და შენარჩუნებულია თვისება (ამ თვისებას უწოდებენ ალგორითმის **ციკლის ინვარიანტს**) - "ციკლის ნებისმიერი იტერაციის წინ A სიმრავლე წარმოადგენს რომელიღაც მინიმალური დამფარავი ხის ქვესიმრავლეს". მორიგ ბიჯზე დამატებული (u, v) წიბო იმგვარად ამოირჩევა, რომ არ დაირღვეს ეს თვისება, ე.ი. $A \cup \{(u, v)\}$ ასევე უნდა იყოს მინიმალური დამფარავი ხის ქვესიმრავლე. ასეთ წიბოს უწოდებენ **უსაფრთხო წიბოს** (safe edge) A-სათვის.

ცხადია, რომ მთავარი პრობლემა მე-3 სტრიქონში უსაფრთხო წიბოს მოძებნაა, მაგრამ მანამდე შევნიშნოთ, რომ ასეთი წიბო გარანტირებულად არსებობს, რადგან მე-4 სტრიქონის შესრულების დროს, ციკლის ინვარიანტის თანახმად, უნდა არსებობდეს მინიმალური დამფარავი ხე T, რომ $A \subseteq T$. ამიტომ უნდა არსებობდეს წიბო $(u, v) \in T$, ისეთი, რომ $(u, v) \notin A$ და (u, v) უსაფრთხო წიბოა A ხისთვის.

ვიდრე უსაფრთხო წიბოს მოძებნის ალგორითმებს განვიხილავდეთ, განვსაზღვროთ რამდენიმე ტერმინი.

$G = (V, E)$ არაორიენტირებული გრაფის $(S, V\backslash S)$ **ჭრილი** (cut) ეწოდება მისი წვეროთა სიმრავლის გაყოფას ორ ქვესიმრავლედ.

Algorithm 6: Generic MST

Input: ბმული არაორიენტირებული გრაფი $G = (V, E)$ და წონის ფუნქცია $w : E \to R$
Output: გრაფის დამფარავი ხე

```
1 GENERIC-MST(G, w) :
2 |  A = ∅;
3 |  while ( A არ არის დამფარავი ხე ) :
4 |  |  მოვძებნოთ (u, v) უსაფრთხო წიბო A ხისთვის;
5 |  |  A = A ∪ {(u, v)};
6 |  return A
```

იტყვიან, რომ $(u, v) \in E$ წიბო **კვეთს** (crosses) $(S, V\backslash S)$ ჭრილს, თუ მისი ერთი ბოლო ეკუთვნის S-ს, ხოლო მეორე ბოლო - $(V\backslash S)$-ს. ჭრილი **შეთანხმებულია წიბოთა A სიმრავლესთან** (respects the set A), თუ A-დან არც ერთი წიბო არ კვეთს ამ ჭრილს. ჭრილის მიერ გადაკვეთილ წიბოთა სიმრავლეში გამოყოფენ უმცირესი წონის წიბოს, რომელსაც **მსუბუქს** (light edges) უწოდებენ.

თეორემა 3.1. ვთქვათ $G = (V, E)$ ბმული არაორიენტირებული გრაფია და მის წიბოთა სიმრავლეზე განსაზღვრულია ნამდვილი წონითი w ფუნქცია. ვთქვათ A წიბოთა სიმრავლეა, რომელიც წარმოადგენს G გრაფის რომელიმე მინიმალური დამფარავი ხის ქვესიმრავლეს. ვთქვათ $(S, V\backslash S)$ წარმოადგენს G გრაფის ისეთ ჭრილს, რომელიც შეთანხმებულია A-სთან, ხოლო (u, v) წიბო ამ ჭრილის მსუბუქი წიბოა. მაშინ (u, v) წიბო წარმოადგენს უსაფრთხო წიბოს A-სათვის.

Proof. ვთქვათ, T მინიმალური დამფარავი ხეა, რომელიც შეიცავს A-ს. დავუშვათ, T არ შეიცავს (u, v) წიბოს, რადგან წინააღმდეგ შემთხვევაში, დასამტკიცებელი დებულება ცხადია. ვაჩვენოთ, რომ არსებობს სხვა მინიმალური დამფარავი ხე T', რომელიც შეიცავს $A \cup \{(u, v)\}$. ამით დამტკიცდება, რომ (u, v) წიბო უსაფრთხოა A-სთვის.
T ბმულია, ამიტომ ის შეიცავს რაიმე p გზას (ერთადერთს) u-დან v-ში. (u, v) წიბო ქმნის ციკლს ამ გზასთან ერთად. რადგან u და v წვეროები ეკუთვნიან $(S, V\backslash S)$ ჭრილის სხვადასხვა ქვესიმრავლეს, p გზაზე არსებობს მინიმუმ ერთი წიბო, რომელიც კვეთს ჭრილს. ვთქვათ, ეს წიბოა (x, y). იგი არ ეკუთვნის A-ს, რადგან ჭრილი შეთანხმებულია A-სთან. დავუმატოთ T ხეს (u, v) წიბო და მიღებული ციკლიდან ამოვიღოთ (x, y) წიბო, მივითებთ ახალ დამფარავ ხეს $T' = T\backslash\{(x, y)\} \cup \{(u, v)\}$.
ახლა, ვაჩვენოთ, რომ T' მინიმალური დამფარავი ხეა. რადგან (u, v) მსუბუქი წიბოა, რომელიც კვეთს $(S, V\backslash S)$ ჭრილს, T-დან ამოჭრილ (x, y) წიბოს, აქვს არანაკლები წონა, ვიდრე მის ნაცვლად დამატებულ (u, v)-ს. ($w(u, v) \leq w(x, y)$). ამიტომ T'-ს წონა შეიძლება მხოლოდ შემცირებულიყო,

$$w(T') = w(T) - w(x, y) + w(u, v) \leq w(T)$$

მაგრამ T მინიმალური დამფარავი ხეა, ე.ი. T'-ც უნდა იყოს იგივე წონის სხვა მინიმალური დამფარავი ხე. ამიტომ T'-ში შემავალი (u, v) წიბო უსაფრთხოა. □

თეორემა 3.1-ის დახმარებით განვიხილოთ როგორ მუშაობს პროცედურა GENERIC-MST. ალგორითმის მუშაობის პროცესში, A სიმრავლე ყოველთვის აციკლურია (ის მინიმალური დამფარავი ხის ქვესიმრავლეა). ალგორითმის მუშაობის ყოველ მომენტში გრაფი $G_A = (V, A)$ წარმოადგენს ტყეს და მისი ნებისმიერი ბმული კომპონენტი არის ხე. (ზოგიერთი ხე შეიძლება შედგებოდეს მხოლოდ ერთი წვეროსგან, მაგ.: როცა ალგორითმი იწყებს მუშაობას, A სიმრავლე ცარიელია, ხოლო ტყე შეიცავს $|V|$ რაოდენობა ერთი წვეროს შემცველ ხეს). გარდა ამისა, A-ს ნებისმიერი უსაფრთხო (u, v) წიბო აკავშირებს G_A-ს სხვადასხვა კომპონენტს, რადგან სიმრავლე $A \cup \{(u, v)\}$ უნდა იყოს აციკლური.
პროცედურა GENERIC-MST-ში, ციკლი 3-5 სტრიქონებში სრულდება $|V| - 1$-ჯერ, რადგან ნაპოვნი უნდა იქნას მინიმალური დამფარავი ხის ყველა $|V| - 1$ წიბო. თავიდან, როცა $A = \varnothing$, G_A-ში არის ხეების $|V|$ რაოდენობა და ყოველი იტერაცია ამცირებს ამ რაოდენობას ერთით. როცა G_A-ში რჩება ერთი ხე, ალგორითმი სრულდება.

შედეგი 3.1. ვთქვათ $G = (V, E)$ ბმული არაორიენტირებული გრაფია და წიბოთა E სიმრავლეზე განსაზღვრულია ნამდვილი წონითი w ფუნქცია. ვთქვათ A წიბოთა სიმრავლეა, რომელიც წარმოადგენს G გრაფის რომელიმე მინიმალური დამფარავი ხის ქვესიმრავლეს. განვიხილოთ ტყე $G_A = (V, A)$ და ვთქვათ $C = (V_C, E_C)$ - მისი ერთ-ერთი ბმული კომპონენტია (ხეა). თუ (u, v) მსუბუქი წიბოა, რომელიც აკავშირებს C-ს G_A-ს რაიმე სხვა კომპონენტთან, მაშინ ეს წიბო უსაფრთხოა A-სთვის.

Proof. ჭრილი $(V_C, V\backslash V_C)$ შეთანხმებულია A-სთან და (u, v) მსუბუქი წიბოა ამ ჭრილისთვის, ამიტომ ის უსაფრთხოა A-სთვის. □

განვიხილოთ მინიმალური დამფარავი ხის პოვნის ორი ალგორითმი. თითოეული მათგანი იყენებს უსაფრთხო წიბოს ამორჩევის საკუთარ წესს (პროცედურა GENERIC-MST, სტრ. 4). **კრასკალის** ალგორითმში A სიმრავლე წარმოადგენს ტყეს, A-ს ემატება უსაფრთხო წიბოები, რომლებიც არიან ორი სხვადასხვა კომპონენტის დამაკავშირებელი მინიმალური წონის მქონე წიბოები. **პრიმის** ალგორითმში A სიმრავლე წარმოადგენს ერთიან ხეს, A-ს ემატება უსაფრთხო წიბოები, რომელთაც აქვთ მინიმალური წონა და რომლებიც აერთიანებენ ფესვის მქონე ხეს ამ ხის გარეთ მქონე წვეროებთან.

3.1 კრასკალის ალგორითმი

კრასკალის ალგორითმის მუშაობის ნებისმიერ მომენტში ამორჩეულ წიბოთა A სიმრავლე (დამფარავი ხის ნაწილი) არ შეიცავს ციკლებს. ალგორითმი ეძებს უსაფრთხო წიბოს ტყეში დასამატებლად, (u, v) მინიმალური წონის მქონე წიბოს პოვნით ყველა იმ წიბოს შორის, რომელიც აკავშირებს სხვადასხვა ხეს ტყეში. აღვნიშნოთ ორი ხე, რომელიც უკავშირდება ერთმანეთს (u, v) წიბოთი C_1-ით და C_2-ით. რადგან (u, v) წიბო მსუბუქია $(C_1, V \backslash C_1)$ ჭრილისთვის, **შედეგი** 3.2-დან გამომდინარეობს, რომ (u, v) წიბო უსაფრთხოა. სურ. 3.2-ზე ნაჩვენებია თუ როგორ მუშაობს ალგორითმი.
ალგორითმი MST-KRUSKAL(G,w) იყენებს სამ ოპერაციას, რომელიც მუშაობს თანაუკვეთ სიმრავლეებზე.
თანაუკვეთ სიმრავლეებზე განსაზღვრული მონაცემთა სტრუქტურა განისაზღვრება თანაუკვეთი დინამიკური სიმრავლეების $S = (S_1, S_2, \ldots, S_k)$ ნაკრებით. ყოველი სიმრავლის იდენტიფიცირება ხდება **წარმომადგენლით** (representative) რომელიც არის ამ სიმრავლის რაიმე ელემენტი. სიმრავლის ყოველი ელემენტი წარმოადგენს რაიმე ობიექტს. აღვნიშნოთ ეს ობიექტი x-ით. განვიხილოთ შემდეგი სამი ოპერაცია:

MAKE-SET(x) ("შევქმნათ სიმრავლე") - ქმნის ახალ სიმრავლეს, რომლის ერთადერთი ელემენტია x (ამიტომ x იქნება წარმომადგენელიც). რადგან სიმრავლეები არ უნდა თანაიკვეთონ, x ობიექტი არ უნდა შედიოდეს არც ერთ სხვა სიმრავლეში.

FIND-SET(x) ("მოვძებნოთ სიმრავლე") - პროცედურა აბრუნებს მიმთითებელს იმ სიმრავლის წარმომადგენელზე, რომელიც შეიცავს x ელემენტს.

UNION(x,y) - აერთიანებს x-ის და y-ის შემცველ დინამიკურ სიმრავლეებს (აღვნიშნოთ ისინი S_x-ით და S_y-ით) ახალ სიმრავლეში. იგულისხმება, რომ ოპერაციის შესრულებამდე აღნიშნული სიმარავლეები არ თანაიკვეთებოდნენ. მიღებული წარმომადგენელი არის $S_x \cup S_y$ სიმრავლის ელემენტი. ხოლო ძველი სიმრავლეები S_x და S_y წაიშლება.

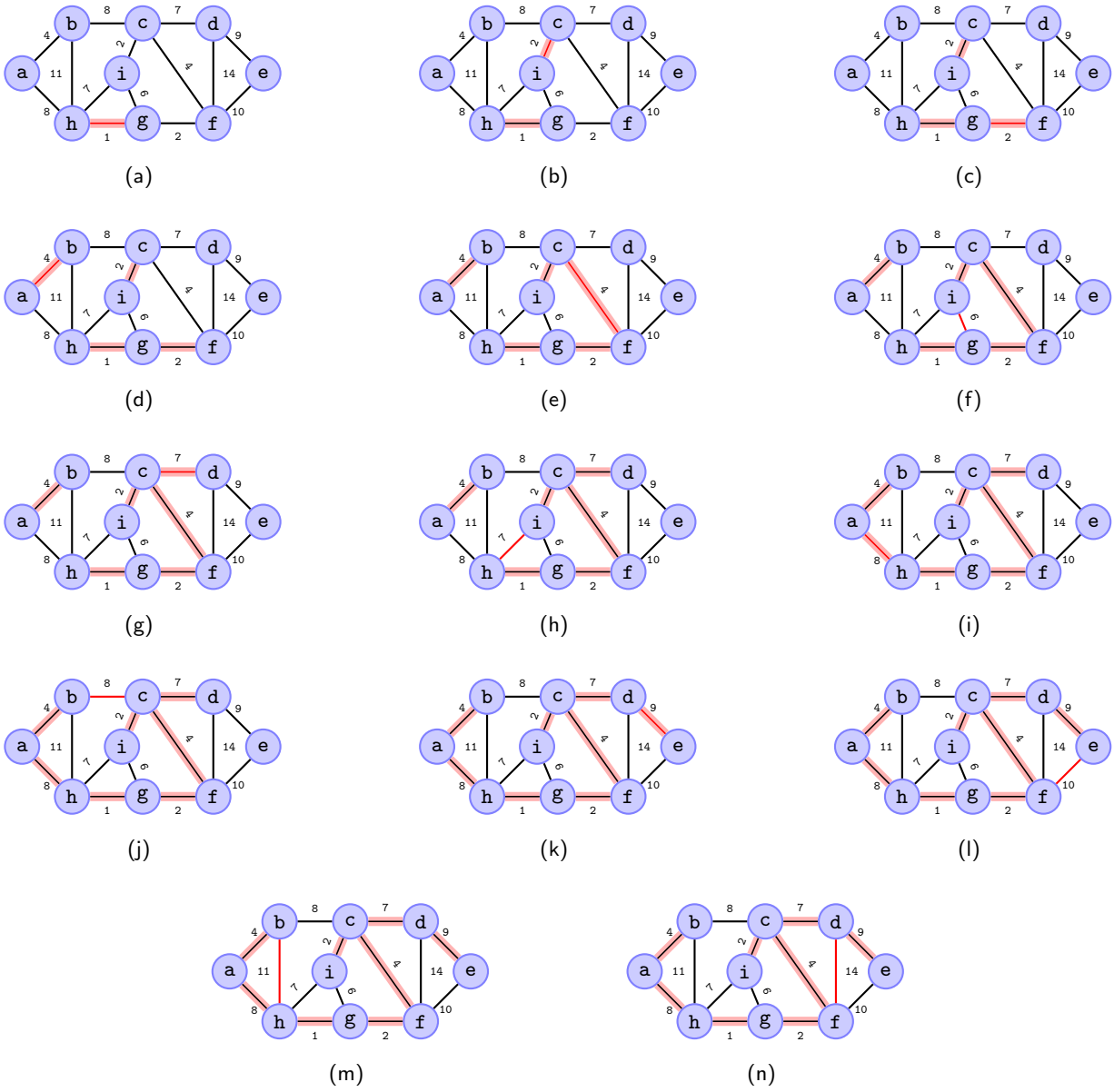
ზემოთ თქმულიდან გამომდინარე, გრაფის ორი u და v წვერო ეკუთვნის ერთ სიმრავლეს (ანუ კომპონენტს), როცა FIND-SET(u) = FIND-SET(v).

Algorithm 7: Minimum Spanning Tree - Kruskal

Input: ბმული არაორიენტირებული გრაფი $G = (V, E)$ და წონის ფუნქცია $w : E \to R$
Output: g რაფის მინიმალური წონის დამფარავი ხე

```
1  MST-KRUSKAL(G, w) :
2      A = ∅;
3      for ∀v ∈ V :
4          MAKE-SET(v);
5      sort(E); // დავალაგოთ წიბოების წონების ზრდის მიხედვით
6      for ∀(u, v) ∈ E :
7          if FIND-SET(u) ≠ FIND-SET(v) :
8              A = A ∪ {(u, v)};
9              UNION(u,v);
10     return A
```

ალგორითმის 2-4 სტრიქონებში ხდება A სიმრავლის ინიციალიზაცია ცარიელი სიმრავლით და იქმნება $|V|$ ხე, რომელიც შეიცავს თითო წვეროს. მე-5 სტრიქონში წიბოები ლაგდება წონათა არაკლებადობის მიხედვით. 6-9 სტრიქონებში ციკლი for ამოწმებს, ეკუთვნიან თუ არა წიბოს წვეროები ერთსა და იმავე ხეს. თუ ეკუთვნიან, მაშინ ამ წიბოს დამატება ტყისათვის არ შეიძლება (რადგან შეიქმნება ციკლი) და ხდება წიბოს უკუგდება, ხოლო თუ წიბოს წვეროები ეკუთვნიან სხვადასხვა ხეს, მაშინ წიბო ემატება A-ს (მე-8 სტრიქონი) და ამ წიბოთი დაკავშირებული ორი ხე ერთიანდება (მე-9 სტრიქონი).



ნახ. 3.2:

კრასკალის ალგორითმის მუშაობის დრო დამოკიდებულია თანაუკვეთ სიმრავლეებზე მონაცემთა სტრუქტურის რეალიზაციაზე. განვიხილოთ თანაუკვეთი სიმრავლეების წარმოდგენები ბმული სიების და ფესვის მქონე ხეების სახით.

თანაუკვეთი სიმრავლეები შეიძლება წარმოდგენილი იქნან ბმული სიების სახით. ყოველ ბმულ სიაში პირველი ობიექტი წარმოადგენს მის წარმომადგენელს. ამ ბმული სიის ყოველი ობიექტი შეიცავს სიმრავლის ელემენტს, მიმთითებელს სიმრავლის შემდეგი ელემენტის შემცველ ობიექტზე და მიმთითებელს წარმომადგენელზე. ყოველ ბმულ სიაში არის მიმთითებელი მის წარმომადგენელზე (head) და მიმთითებელი მის ბოლო ელემენტზე (tail).

თანაუკვეთი სიმრავლეების ასეთი წარმოდგენის შემთხვევაში, MAKE-SET(x) და FIND-SET(x) პროცედურებს სჭირდება $O(1)$ დრო. MAKE-SET(x) ქმნის ახალ ბმულ სიას ერთადერთი x ობიექტით, ხოლო FIND-SET(x) აბრუნებს მიმთითებელს x ელემენტის შემცველი სიმრავლის წარმომადგენელზე. UNION(x,y) პროცედურა სრულდება შემდეგნაირად: სია, რომელიც შეიცავს x ელემენტს, ემატება სიას, რომელიც შეიცავს y ელემენტს. y ელემენტის შემცველი სიის tail მიმთითებელი გამოიყენება იმისთვის, რომ სწრაფად განვსაზღვროთ სად დავამატოთ x-ის შემცველი სია. ახალი სიმრავლის მიმთითებელი ხდება y-ის შემცველი სიის მიმთითებელი. ამასთან უნდა განახლდეს მიმთითებლები x-ის შემცველი სიის ყოველი ობიექტისთვის, რაზეც დახარჯული დრო წრფივადაა დამოკიდებული x-ის შემცველი სიის სიგრძეზე.

ვთქვათ, გვაქვს $x_1, \ldots, x_n$ ობიექტი. სრულდება MAKE-SET(x)-ის n ოპერაცია და, შემდეგ, UNION(x,y)-ის $n-1$ ოპერაცია. MAKE-SET(x)-ის n ოპერაციას სჭირდება $\Theta(n)$ დრო, ხოლო, რადგან UNION(x,y)-ის i-ური ოპერაცია აახლებს i ობიექტს, UNION(x,y)-ის $n-1$ ოპერაციით განახლებული ობიექტების რაოდენობაა:

$$\sum_{i=1}^{n-1} i = \Theta(n^2)$$

ოპერაციების მიმდევრობა ბმული სის შემთხვევაში	
ოპერაცია	განახლებული ობიექტების რაოდენობა
MAKE-SET(x_1)	1
MAKE-SET(x_2)	1
⋮	⋮
MAKE-SET(x_n)	1
UNION(x_1, x_2)	1
UNION(x_2, x_3)	2
UNION(x_3, x_4)	3
⋮	⋮
UNION(x_{n-1}, x_n)	$n-1$

ოპერაციების საერთო რაოდენობააა $2n-1$, ასე რომ საშუალოდ თითოეულ ოპერაციას სჭირდება $\Theta(n)$ დრო.

UNION პროცედურას, წარმოდგენილი რეალიზაციით, უარეს შემთხვევაში, ერთი გამოძახებისთვის, საშუალოდ, სჭირდება $\Theta(n)$ დრო, რადგანაც შეიძლება აღმოჩნდეს, რომ გრძელ სიას ვაერთებთ მოკლე სიასთან და, ამიტომ უნდა განახლდეს ამ გრძელი სიის ყველა წევრის მიმთითებელი წარმომადგენელზე. დავუშვათ, ახლა, რომ ყოველი სია შეიცავს მისი სიგრძის აღმნიშვნელ ველს და ყოველთვის ვაერთებთ მოკლე სიას გრძელთან, მაშინ მტკიცდება, რომ MAKE-SET, FIND-SET, UNION m ოპერაციების მიმდევრობის შესრულებისთვის (რომელთაგანაც n მიმდევრობა MAKE-SET ოპერაციაა) საჭირო დრო ხდება $O(m + n \log n)$.

თუ თანაუკვეთ სიმრავლეებს წარმოვადგენთ ფესვის მქონე ხეების სახით, სადაც ყოველი კვანძი სიმრავლის ერთ ელემენტს შეესაბამება, ხოლო ყოველი ხე წარმოადგენს ერთ სიმრავლეს, მაშინ შესაძლებელია MAKE-SET, FIND-SET, UNION პროცედურების უფრო სწრაფი განხორციელება.

თანაუკვეთ სიმრავლეთა ტყეში (disjoint-set forest) ყოველი ელემენტი მიუთითებს მხოლოდ მშობელზე. ყოველი ხის ფესვი არის წარმომადგენელი და არის თავისი თავის მშობელიც.

თანაუკვეთ სიმრავლეებზე ოპერაციების ასიმტოტურად სწრაფად შესრულების საშუალებას იძლევა ორი ევრისტიკა: " **გაერთიანება რანგით**" და " **გზის შეკუმშვა**".

ოპერაციები შემდეგნაირად სრულდება:

MAKE-SET(x) - ქმნის ხეს ერთი კვანძით $\Theta(1)$ დროში, n რაოდენობა ერთ ელემენტიანი სიმრავლის შექმნას დასჭირდება $\Theta(n)$ დრო.

FIND-SET(x) - აბრუნებს x კვანძის შემცველი ხის x კვანძიდან ხის ფესვამდე გადაადგილებით (რომელიც არის წარმომადგენელი) მშობლების მიმთითებლის გავლით. თითოეულ ასეთ ოპერაციას სჭირდება $O(n)$ დრო. გავლილი კვანძები ამ გზაზე ქმნიან ძებნის გზას (find path).

UNION(x,y) - ხორციელდება y ხის ფესვის მიერთებით x ხის ფესვთან და y ხის ამოღებით ტყიდან. ამ ოპერაციას სჭირდება $\Theta(1)$ დრო.

გავეცნოთ ორ ევრისტიკას, რომელთა გამოყენებითაც შესაძლებელია დროითი საზღვრის შემცირება:

I **ევრისტიკა: გაერთიანება რანგით** (union by rank) დაფუძნებულია იდეაზე, რომ ყოველთვის, UNION ოპერაციის შესრულების დროს, ნაკლები რაოდენობის კვანძის შემცველი ხის ფესვი უერთდებოდეს მეტი რაოდენობის კვანძის შემცველი ხის ფესვს. ამისთვის გამოიყენება ფესვის რანგის (rank) ცნება. rank[x] არის x კვანძის სიმაღლე (x-დან მის შთამომავალ ფოთლამდე წიბოების რაოდენობა ყველაზე გრძელ გზაზე). ამ ევრისტიკის განსახორციელებლად, ყოველი კვანძისთვის შენახული უნდა გვქონდეს rank[x] (MAKE-SET პროცედურის მიერ ერთელემენტიანი სიმრავლის შექმნის დროს, rank[x]=0). FIND-SET არ ცვლის რანგებს. ადვილი დასამტკიცებელია, რომ რადგან ნებისმიერი კვანძის რანგი არ აღემატება $\lfloor \log n \rfloor$-ს, ძებნის ყოველი ოპერაცია ხორციელდება $O(\log n)$ დროში, ამრიგად, არა უმეტეს $n-1$ რაოდენობა UNION ოპერაციისა და m რაოდენობა FIND-SET ოპერაციის შესრულებას დასჭირდება $O(n + m \log n)$ დრო.

II **ევრისტიკა: გზის შეკუმშვა** (path compression) გამოიყენება FIND-SET ოპერაციის შესრულების პროცესში და ყველა კვანძს უშუალოდ უთითებს ფესვზე. ეს ევრისტიკა არ ცვლის კვანძების რანგებს.

ორი ხის გაერთიანების UNION პროცედურის გამოყენების დროს გვაქვს ორი შემთხვევა:

1. თუ ორ ხეს აქვს ერთნაირი რანგი, მაშინ ვირჩევთ ერთ-ერთი ხის ფესვს მშობლად და ვზრდით მის რანგს ერთით.
2. თუ ერთი ხის რანგი მეტია მეორე ხის რანგზე, მაშინ დიდი რანგის მქონე ხის ფესვი ხდება მშობელი კვანძი ნაკლები რანგის მქონე ხის ფესვისთვის, ხოლო რანგების მნიშვნელობები რჩება იგივე.

შემდეგ ფსევდოკოდებში $p[x]$ აღნიშნავს x-ს მშობელს.

Algorithm 8: Disjoint Set Data Structure Operations

```
1  MAKE-SET(x) :
2  |  p[x] = x;
3  |  rank[x] = 0;
4  FIND-SET(x) :
5  |  if x ≠ p[x] :
6  |  |  p[x] = FIND-SET(p[x]);
7  |  return p[x];
8  LINK(x,y) :
9  |  if rank[x] > rank[y] :
10 |  |  p[y] = x;
11 |  else:
12 |  |  p[x] = y;
13 |  |  if rank[x] == rank[y] :
14 |  |  |  rank[y] += 1;
15 UNION(x,y) :
16 |  LINK(FIND-SET(x), FIND-SET(y));
```

დავითვალოთ კრასკალის ალგორითმის მუშაობის დრო. ინიციალიზაციას სჭირდება დრო $O(V)$ (სტრ. 2-4). E წიბოების დალაგებას წონების მიხედვით - $O(E \log E)$ (სტრ. 5). 6-9 სტრიქონებში სრულდება $O(E)$ რაოდენობა FIND-SET და UNION ოპერაცია თანაუკვეთ სიმრავლეთა ტყეზე. V რაოდენობა MAKE-SET ოპერაციასთან ერთად, ამას სჭირდება $O((V+E)\alpha(V))$ დრო, სადაც α ძალიან ნელა ზრდადი ფუნქციაა. რადგან G ბმული გრაფია, სამართლიანია $|E| \geq |V|-1$ ასე, რომ, ოპერაციები თანაუკვეთ სიმრავლეებზე საჭიროებენ $O(E\alpha(V))$ დროს, გარდა ამისა, რადგან $\alpha(|V|) = O(\log V) = O(\log E)$, კრასკალის ალგორითმის მუშაობის დროა $O(E \log E)$. შევნიშნოთ, რომ $|E| < |V|^2$, ამიტომ $\log |E| = O(\log V)$ და კრასკალის ალგორითმის მუშაობის დრო შეიძლება ჩავწეროთ როგორც $O(E \log V)$.

3.2 პრიმის ალგორითმი

პრიმის ალგორითმი გამოირჩევა იმით, რომ A სიმრავლის წიბოები ყოველთვის ქმნიან ერთიან ხეს. პრიმის ალგორითმით მინიმალური დამფარავი ხის ფორმირება იწყება ნებისმიერი საწყისი r წვეროდან. ყოველ ბიჯზე ხეს

ემატება უმცირესი წონის წიბო იმ წიბოებს შორის, რომლებიც წვეროს აერთებენ ხის არაწევრ წვეროებთან. ზემოხსენებული შედეგის მიხედვით ასეთი წიბო უსაფრთხოა A-სათვის, ე.ი. მიიღება მინიმალური დამფარავი ხე.
პროცედურისათვის შემავალი მონაცემებია: ბმული G გრაფი, წიბოთა w წონები და r საწყისი წვერო. რეალიზაციისას მნიშვნელოვანია სწრაფად ავარჩიოთ მსუბუქი წიბო. ალგორითმის მუშაობისას ყველა წვერო, რომელიც ჯერ არ გამხდარა ხის წევრი, ინახება Q პრიორიტეტებიან რიგში. v წვეროს პრიორიტეტი განისაზღვრება $key[v]$ მნიშვნელობით, რომელიც უდრის იმ წიბოების მინიმალურ წონას, რომელიც აერთებს v-ს A ხესთან. თუ ასეთი წიბო არ არსებობს - $key[v] = \infty$. $\pi[v]$ ველი ხის წვეროებისთვის მიუთითებს მშობელს, ხოლო სხვა წვეროებისათვის ხის წვეროს, რომლისკენაც მივყავართ $key[v]$ წონის წიბოს (თუ ასეთი წიბო რამდენიმეა, მაშინ მიეთითება ერთ-ერთი).
ალგორითმის მუშაობის პროცესში, A სიმრავლე პროცედურა GENERIC-MST-ში არის შემდეგი:

$$A = \{(v, \pi[v]) : v \in V \backslash \{r\} \backslash Q\}$$

როცა ალგორითმი ასრულებს მუშაობას, Q პრიორიტეტებიანი რიგი ცარიელია, ხოლო მინიმალური დამფარავი ხე G-თვის არის ხე:

$$A = \{(v, \pi[v]) : v \in V \backslash \{r\}\}$$

Algorithm 9: Minimum Spanning Tree - Prim

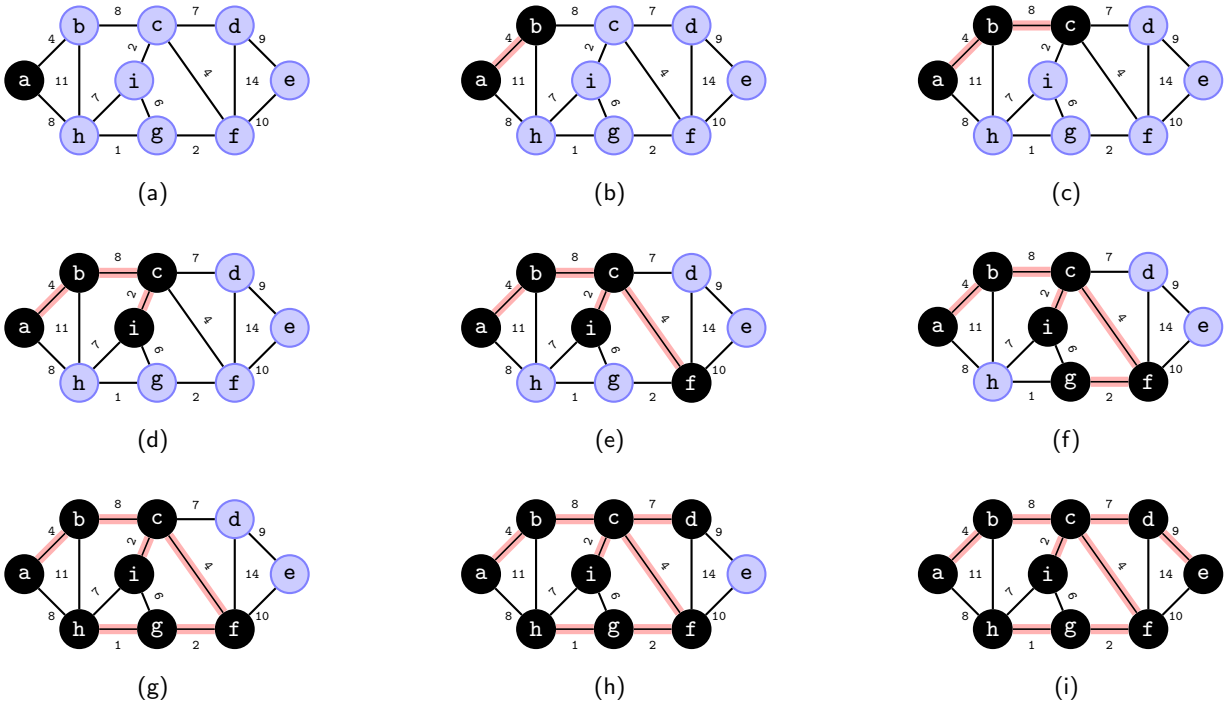
Input: ბმული არაორიენტირებული გრაფი $G = (V, E)$, წონის ფუნქცია $w : E \to R$ და საწყისი წვერო r
Output: გრაფის მინიმალური წონის დამფარავი ხე

```
1  MST-PRIM(G, w, r) :
2  |  for ∀u ∈ V :
3  |  |  key[u] = ∞;
4  |  |  π[u] = NIL;
5  |  key[r] = 0;
6  |  Q = V; // პრიორიტეტული რიგი, კეყ გვაძლევს პრიორიტეტს
7  |  while Q ≠ ∅ :
8  |  |  u = EXTRACT-MIN(Q);
9  |  |  for ∀v ∈ adj[u] :
10 |  |  |  if v ∈ Q and w(u,v) < key[v] :
11 |  |  |  |  π[v] = u;
12 |  |  |  |  key[v] = w(u,v);
13 |  return π; // ხე განისაზღვრება (v, π[v]) წიბოებით
```

2-6 სტრიქონებში ყველა წვეროს გასაღები ხდება ∞-ს ტოლი, გარდა r ფესვისა, რომლის გასაღები 0-ს ტოლია და ის აღმოჩნდება პირველი დასამუშავებელი წვერო. ყველა წვეროსთვის მშობლების ველში ჩაიწერება მნიშვნელობა NIL და ყველა წვერო ჩაიწერება Q პრიორიტეტებიან რიგში. while ციკლის ყოველი იტერაციის წინ, 7-12 სტრიქონებში:

- $A = \{(v, \pi[v]) : v \in V \backslash \{r\} \backslash Q\}$
- მინიმალურ დამფარავ ხეში უკვე შესული წვეროები ეკუთვნის $V \backslash Q$ სიმრავლეს
- ყველა $v \in Q$ წვეროსთვის, სამართლიანია: თუ $\pi[v] \neq NIL$, მაშინ $key[v] < \infty$ და $key[v]$ არის იმ $(v, \pi[v])$ მსუბუქი წიბოს წონა, რომელიც აკავშირებს v-ს რაიმე წვეროსთან, რომელიც უკვე მოთავსებულია მინიმალურ დამფარავ ხეში.

მე-8 სტრიქონში განისაზღვრება u წვერო, რომელიც ეკუთვნის $(V \backslash Q, Q)$ ჭრილის გადამკვეთ მსუბუქ წიბოს (პირველი იტერაციის გარდა). ხდება u-ს ამოღება Q-დან, და მისი დამატება ხის წვეროების $V \backslash Q$ სიმრავლეში. მოცემული ციკლის პირველი გავლისას ხე შედგება ერთადერთი წვეროსაგან. ყველა დანარჩენი წვერო იმყოფება რიგში. key[v]-ს მნიშვნელობა მათთვის r-დან v-ში წიბოს სიგრძის ან უსასრულობის (თუკი წიბო არ არსებობს) ტოლია. ალგორითმის მუშაობა მოცემულია სურ. 3.3-ზე. EXTRACT-MIN(Q) შლის მინიმალურ ელემენტს Q რიგიდან და აბრუნებს მას.
ალგორითმის მუშაობის დრო დამოკიდებულია Q პრიორიტეტებიანი რიგის რეალიზაციაზე. თუკი გამოყენებულია ორობითი გროვა, მაშინ მუშაობის დრო კრასკალის ალგორითმის ანალოგიურია - $O(E \log V)$. (2-6 სტრიქონებში ინიციალიზაცია შეიძლება შევასრულოთ $O(V)$ დროში. while ციკლი სრულდება $|V|$-ჯერ და ყოველი ოპერაცია EXTRACT-MIN სრულდება $O(\log V)$ დროში. EXTRACT-MIN-ის შესრულების საერთო დრო გახდება $O(V \log V)$. ციკლი 9-12 სტრიქონებში სრულდება $O(E)$-ჯერ. მე-9 სტრიქონში შემოწმება - $O(1)$ დროში. ხოლო მე-12 სტრიქონი



ნახ. 3.3:

შესრულდება $O(\log V)$ დროში. საბოლოოდ, პრიმის ალგორითმის მუშაობის საერთო დრო იქნება $O(V \log V + E \log V) = O(E \log V)$. ფიბონაჩის გროვის გამოყენების შემთხვევაში შეფასება შეიძლება შემცირდეს $O(E + V \log V)$-მდე.

3.3 სავარჯიშოები

1. იპოვეთ მინიმალური დამფარავი ხე სურ. 3.4 გრაფებისთვის:
   (ა) კრასკალის ალგორითმით
   (ბ) პრიმის ალგორითმით (c საწყისი წვეროდან)

ნახ. 3.4:

2. აჩვენეთ, რომ გრაფს აქვს ერთადერთი მინიმალური დამფარავი ხე, თუ გრაფის ყოველი ჭრილისთვის არსებობს ერთადერთი მსუბუქი წიბო, რომელიც კვეთს ამ ჭრილს. მოიყვანეთ კონტრმაგალითი, რომელიც აჩვენებს, რომ საწინააღმდეგო დებულება არ სრულდება.

3. ვთქვათ, (u, v) მინიმალური წონის მქონე წიბოა G გრაფში, აჩვენეთ, რომ (u, v) ეკუთვნის G გრაფის რომელიმე მინიმალურ დამფარავ ხეს.

4. $G(V, E)$ ბმული, არაორიენტირებული გრაფისთვის, განვიხილოთ წიბოების E სიმრავლე, რომლის ყოველი ელემენტი წარმოადგენს მსუბუქ წიბოს გრაფის რაიმე შესაძლო ჭრილისთვის. მოიყვანეთ მაგალითი, როცა ეს სიმრავლე არ ქმნის მინიმალურ დამფარავ ხეს.

$\alpha(n)$ ძალიან ნელა ზრდადი ფუნქციაა. რეალურ ამოცანებში, რომელშიც გამოიყენება თანაუკვეთი სიმრავლეები, $\alpha(n) \leq 4$.

თავი 4

უმოკლესი გზები ერთი წვეროდან

თუ მოცემულია შეწონილი გრაფი, ხშირად საჭიროა ხოლმე მის ორ წვეროს შორის უმოკლესი გზის დადგენა (ორ წვეროს შემაერთებელ ყველა შესაძლო გზას შორის ისეთის არჩევა, რომლის წიბოთა წონების ჯამი მინიმალურია).
ამ ამოცანის გადაჭრაზე ძალიან ბევრი სხვა ამოცანაა დამოკიდებული, მათ შორის:

- სატრანსპორტო ქსელებში ორ პუნქტს შორის უმოკლესი გზის პოვნა: თუ გრაფს განვიხილავთ როგორც ქალაქებს (წვეროები) და მათ შემაერთებელ გზებს (წიბოები), ან ქალაქში ქუჩებს (წიბოები) და მათ გადაკვეთებს (წვეროები), ერთი პუნქტიდან მეორეში გადასვლისათვის უმცირესი გზის გამოთვლა ამ ამოცანის გადაჭრით შეიძლება;

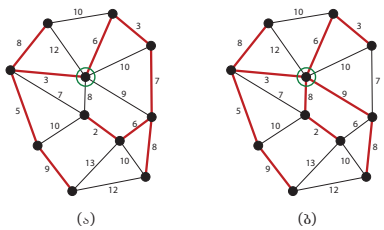
- ნალაპარაკევი ტექსტის ამოცნობის ერთ-ერთი უმთავრესი ამოცანა ერთნაირი ჟღერადობის სიტყვების (ომოფონების) განსხვავებაა. ასეთ სიტყვებზეა აგებული აკაკი წერეთლის ცნობილი ლექსი „აღმართ-აღმართ”:

  აღმართ-აღმართ მივდიოდი ***მე ნელა***,
  სერზედ შევდექ, ჭმუნვის ალი ***მენელა***;
  მზემან სხივი მომაფინა ***მა შინა***,
  სიცოცხლე ვგრძენ, სიკვდილმა ***ვერ მა შინა***.

  თუ ენის სიტყვებს აღვნიშნავთ, როგორც გრაფის წვეროებს და „მსგავს” სიტყვებს წიბოებით შევაერთებთ (თანაც მსგავსების კოეფიციენტს წიბოს წონად მივუწერთ - რაც უფრო მსგავსია ორი სიტყვა, უფრო ნაკლებს), წვეროებს შორის უმოკლესი გზის პოვნა წინადადების აზრის დადგენაში დაგვეხმარება.

- გრაფთა განლაგებაში: ხშირად საჭიროა ხოლმე გრაფის „ცენტრის” დადგენა და ისე განლაგება, რომ იგი მის შუაგულში მოექცეს. ასეთი შეიძლება იყოს წვერო, რომლის მაქსიმალური დაშორება ყველა სხვა წვეროსთან ყველაზე დაბალია. ცხადია, რომ ამის დასადგენად საჭიროა ნებისმიერ ორ წვეროს შორის მანძილის ცოდნა.

აღსანიშნავია, რომ უმცირესი დამფარავი ხე ყოველთვის უმცირეს მანძილს არ მოგვცემს, როგორც ეს შემდგომ ნახაზშია ნაჩვენები.



ნახ. 4.1: მინიმალური დამფარავი ხე (ა) და შემოხაზული წვეროდან უმოკლესი მანძილის ხე (ბ)

საგარჯიშო 4.1: მოიყვანეთ სხვა ხეების მაგალითი, რომლებშიც უმცირესი დამფარავი ხე არ მოგვცემს უმოკლეს მანძილებს.

4.1 უმოკლესი გზის პოვნის ამოცანა

ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი ნამდვილი წონითი $w : E \to R$ ფუნქციით. $p =< v_0, v_1, \ldots, v_k >$ გზის **წონას** (weight) უწოდებენ ამ გზაში შემავალი ყველა წიბოს წონების ჯამს:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

u-დან v-ში **უმოკლესი გზის წონა** (shortest-paths weight), განსაზღვრების თანახმად, ტოლია:

$$\delta(u, v) = \begin{cases} \min\{w(p)\} & \text{თუ არსებობს გზა } u\text{-დან } v\text{-ში} \\ \infty & \text{წინააღმდეგ შემთხვევაში} \end{cases}$$

უმოკლესი გზა (shortest-path) u-დან v-ში - ესაა ნებისმიერი p გზა u-დან v-ში, რომლისთვისაც $w(p) = \delta(u, v)$. წონებში შეიძლება ვიგულისხმოთ არა მარტო მანძილები, არამედ დრო, ღირებულება, ჯარიმა, ზარალი და ა.შ. სიგანეში ძებნის ალგორითმი შეგვიძლია განვიხილოთ, როგორც უმოკლესი გზების შესახებ ამოცანის ამოხსნის კერძო შემთხვევა, როცა თითოეული წიბოს წონა 1-ის ტოლია.

განიხილავენ უმოკლესი გზების ამოცანის სხვადასხვა ვარიანტს. ამ თავში ჩვენ განვიხილავთ ამოცანას **უმოკლესი გზების შესახებ ერთი წვეროდან** (single-source shortest-path problem): მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი და საწყისი წვერო s (source vertex). საჭიროა ვიპოვოთ უმოკლესი გზები s-დან ყველა $v \in V$ წვერომდე. ამ ამოცანის ამოხსნის ალგორითმი გამოიყენება სხვა ამოცანების ამოსახსნელადაც, კერძოდ:

უმოკლესი გზა ერთი წვეროსაკენ: მოცემულია საბოლოო t წვერო (destination vertex). საჭიროა მოვძებნოთ უმოკლესი გზები t წვერომდე ყოველი $v \in V$ წვეროდან. თუკი შევაბრუნებთ მიმართულებას ყველა წიბოზე, ეს ამოცანა დაიყვანება ამოცანაზე უმოკლესი გზების შესახებ ერთი წვეროდან.

უმოკლესი გზა წვეროთა მოცემული წყვილისათვის: მოცემულია u და v წვეროები, მოვძებნოთ უმოკლესი გზა u-დან v-ში. რა თქმა უნდა, თუკი ჩვენ მოვძებნით ყველა უმოკლეს გზას u-დან, ამოცანა ამოიხსნება. უნდა აღინიშნოს, რომ უფრო სწრაფი მეთოდი (რომელიც გამოიყენებდა იმ ფაქტს, რომ უმოკლესი გზა მხოლოდ ორ წვეროს შორისაა მოსაძებნი) ჯერჯერობით ნაპოვნი არ არის.

უმოკლესი გზები წვეროთა ყველა წყვილისათვის: წვეროთა ყოველი u და v წყვილისათვის მოვძებნოთ უმოკლესი გზა u-დან v-ში. ამ ამოცანის ამოხსნა შეიძლება, თუკი რიგ-რიგობით ვიპოვით უმოკლეს გზას წვეროთა ყველა წყვილისათვის. თუმცა ეს არაა ოპტიმალური მეთოდი და უფრო ეფექტურ მიდგომას მომდევნო თავებში განვიხილავთ.

4.1.1 უმოკლესი გზების ამოცანის ოპტიმალური სტრუქტურა

უმოკლესი გზების პოვნის ალგორითმები, ჩვეულებრივ, ეყრდნობიან იმ თვისებას, რომ უმოკლესი გზის ყოველი ნაწილი თვითონ არის უმოკლესი გზა. ე.ი. უმოკლესი გზების ამოცანას აქვს ოპტიმალურობის თვისება ქვეამოცანებისათვის, რაც იმას ნიშნავს, რომ ამოცანის ამოსახსნელად შესაძლოა გამოყენებულ იქნას დინამიკური პროგრამირების მეთოდი ან ხარბი ალგორითმი. მართლაც, დეიქსტრას ალგორითმი ხარბ ალგორითმს წარმოადგენს, ხოლო ფლოიდ-ვორშელის ალგორითმი, რომელიც წვეროთა ყველა წყვილისთვის ეძებს უმოკლეს გზებს, დინამიკური პროგრამირების მეთოდს იყენებს.

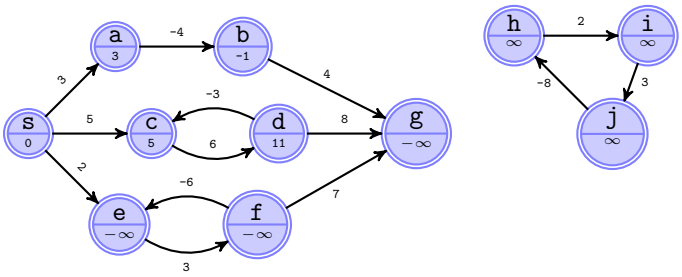
ლემა 4.1. (უმოკლესი გზის მონაკვეთები უმოკლესია). ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით. თუ $p =< v_1, v_2, \ldots, v_k >$ უმოკლესი გზაა v_1-დან v_k-მდე და $1 \le i \le j \le k$, მაშინ $p_{ij} =< v_i, v_{i+1}, \ldots, v_j >$ წარმოადგენს უმოკლეს გზას v_i-დან v_j-მდე.

Proof. თუ p გზას დავშლით შემადგენელ ნაწილებად, $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$, შესრულდება შემდეგი: $w(p) = w(p_{1i}) + w(p_{ij}) + w(p_{jk})$. ახლა დავუშვათ, რომ არსებობს გზა p'_{ij} v_i-დან v_j-მდე, რომლის წონაც აკმაყოფილებს უტოლობას: $w(p'_{ij}) < w(p_{ij})$. მაშინ $v_1 \overset{p_{1i}}{\rightsquigarrow} v_i \overset{p'_{ij}}{\rightsquigarrow} v_j \overset{p_{jk}}{\rightsquigarrow} v_k$ არის გზა v_1-დან v_k-მდე, რომლის წონა $w(p) = w(p_{1i}) + w(p'_{ij}) + w(p_{jk})$ ნაკლებია $w(p)$-ზე, რაც ეწინააღმდეგება პირობას, რომ p უმოკლესი გზაა v_i-დან v_j-მდე.

(ჩავთვალოთ, რომ ნებისმიერი ნამდვილი $a \neq -\infty$ რიცხვისთვის სრულდება: $a + \infty = \infty + a = \infty$ და ნებისმიერი ნამდვილი $a \neq \infty$ რიცხვისთვის სრულდება: $a + (-\infty) = (-\infty) + a = -\infty$) □

4.1.2 უარყოფითი წონის მქონე წიბოები

ზოგიერთ შემთხვევაში, წიბოთა წონები შესაძლოა უარყოფითი იყოს. ამ დროს დიდი მნიშვნელობა აქვს არსებობს თუ არა უარყოფითწონიანი ციკლი. თუ გრაფი არ შეიცავს s წვეროდან მიღწევად უარყოფითწონიან ციკლს, მაშინ ყოველი $v \in V$ წვეროსთვის $\delta(s, v)$ არის სასრული სიდიდე. ხოლო თუ s წვეროდან შესაძლებელია უარყოფითწონიან ციკლთან მისვლა, მაშინ არც ერთი გზა s წვეროდან ციკლის წვერომდე არ იქნება უმოკლესი, რადგან შეგვიძლია წონის განუწყვეტლივ შემცირება. ამრიგად, ასეთ შემთხვევაში უმოკლესი გზა არ არსებობს და თვლიან, რომ $\delta(s, v) = -\infty$.



ნახ. 4.2:

სურ. 4.2-ზე ნაჩვენებია რა გავლენას ახდენს უარყოფითწონიანი წიბოები და ციკლები უმოკლესი გზების წონებზე: რადგან, s წვეროდან a და b წვეროებამდე ერთადერთი გზა არსებობს, ამიტომ:

$$\delta(s, a) = w(s, a) = 3 \qquad \delta(s, b) = w(s, a) + w(a, b) = -1$$

s წვეროდან c წვერომდე უამრავი გზა არსებობს: $< s, c >, < s, c, d, c >, < s, c, d, c, d, c >$ და ა.შ., მაგრამ, რადგან $< c, d, c >$ ციკლის წონა დადებითია, უმოკლესი გზა s წვეროდან c წვერომდე არის $< s, c >$ და მისი წონაა $\delta(s, c) = 5$, ანალოგიურად, $\delta(s, d) = 11$.
s წვეროდან e წვერომდეც უამრავი გზა არსებობს: $< s, e >, < s, e, f, e >, < s, e, f, e, f, e >$ და ა.შ., მაგრამ, რადგან $< e, f, e >$ ციკლის წონა უარყოფითია, არ არსებობს უმოკლესი გზა s წვეროდან e წვერომდე და $\delta(s, e) = -\infty$, ანალოგიურად, $\delta(s, f) = -\infty$. რადგან g წვერო მიღწევადია f-დან, შეიძლება მოიძებნოს უსასრულოდ დიდი უარყოფითი წონის მქონე გზები s წვეროდან g წვერომდე, ამიტომ $\delta(s, g) = -\infty$.
h, i, j წვეროებიც ქმნიან უარყოფითწონიან ციკლს, მაგრამ ისინი არ არიან მიღწევადი s წვეროდან და ამიტომ $\delta(s, h) = \delta(s, i) = \delta(s, j) = \infty$.

4.1.3 ციკლები

შეიძლება თუ არა უმოკლესი გზა შეიცავდეს ციკლს? დავრწმუნდით, რომ უმოკლესი გზა არ შეიძლება შეიცავდეს უარყოფითწონიან ციკლს. ის არ შეიძლება შეიცავდეს დადებითწონიან ციკლსაც, რადგან ამ ციკლის ამოღებით უმოკლესი გზიდან, მივიღებთ გზას საწყისი წვეროდან იმავე წვერომდე, რომელსაც აქვს ნაკლები წონა. თუ ციკლის წონა ნულია, მაშინ მისი ამოღებით უმოკლესი გზიდან, მივიღებთ გზას საწყისი წვეროდან იმავე წვერომდე, რომელსაც აქვს იგივე წონა და რომელიც არ შეიცავს ციკლს. ამიტომ, ზოგადობის შეუზღუდავად შეიძლება ჩავთვალოთ, რომ თუ ვეძებთ უმოკლეს გზებს, ისინი არ შეიცავენ ციკლებს. რადგან $G = (V, E)$ გრაფის ნებისმიერ აციკლურ გზაში შეიძლება შედიოდეს არაუმეტეს $|V|$ რაოდენობა წვეროებისა და $|V| - 1$ რაოდენობა წიბოებისა, შეიძლება შემოვიფარგლოთ უმოკლესი გზების პოვნით, რომელიც შეიცავს წიბოების არაუმეტეს $|V| - 1$ რაოდენობას.

4.1.4 უმოკლესი გზების წარმოდგენა

ზოგჯერ საჭირო ხდება არა მარტო უმოკლესი გზის წონის გამოთვლა, არამედ თავად ამ გზის დადგენაც. ასეთ შემთხვევაში იყენებენ იმავე მეთოდს, რომლითაც პოულობდნენ გზას სიგანეში ძებნის ხეებში. მოცემულ $G = (V, E)$ გრაფში, ყოველი $v \in V$ წვეროსთვის გამოითვლება $\pi[v]$ ატრიბუტის, მშობლის, **წინამორბედის** (predecessor) მნიშვნელობა, რომლის როლში გამოდის ან სხვა წვერო, ან მნიშვნელობა NIL. ამ თავში განხილული ალგორითმებით $\pi[v]$ ატრიბუტები ისე გამოითვლება, რომ v წვეროში დაწყებული წინამორბედების ჯაჭვი, გვაძლევს საშუალებას ავაგოთ s წვეროდან v წვეროში გამავალი უმოკლესი გზის შებრუნებული გზა. $G_\pi = (V_\pi, E_\pi)$

ქვეგრაფს უწოდებენ **წინამორბედობის ქვეგრაფს** (predecessor subgraph), სადაც

$$V_\pi = \{v \in V : \pi[v] \neq NIL\} \cup \{s\} \qquad E_\pi = \{(\pi[v], v) \in E : v \in V_\pi \backslash \{s\}\}$$

უმოკლესი გზის პოვნის ალგორითმების დასრულების შემდეგ, G_π წარმოადგენს "უმოკლესი გზების ხეს".
ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით. ვთქვათ, გრაფი არ შეიცავს $s \in V$ საწყისი წვეროდან მიღწევად უარყოფითწონიან ციკლებს. s ფესვის მქონე **უმოკლესი გზების ხე** (shortest-paths tree) არის $G' = (V', E')$ ორიენტირებული ქვეგრაფი, სადაც $V' \subseteq V$, $E' \subseteq E$ განისაზღვრება შემდეგი პირობებით:

1. V' - არის წვეროთა სიმრავლე, რომელიც მიღწევადია $s \in V$ საწყისი წვეროდან G გრაფში.
2. G' გრაფი წარმოადგენს ხეს s წვეროს მქონე ფესვით.
3. ყოველი $v \in V'$ წვეროსთვის ცალსახად განსაზღვრული მარტივი გზა s წვეროდან v წვეროში G' გრაფში, ემთხვევა უმოკლეს გზას s წვეროდან v წვეროში G გრაფში.

4.1.5 რელაქსაცია

რელაქსაციის მექანიზმი ასეთია: თითოეული $v \in V$-სათვის ვინახავთ რაღაც $d[v]$ რიცხვს, ატრიბუტს, რომელიც წარმოადგენს s-დან v-ში უმოკლესი გზის წონის ზედა შეფასებას, ანუ, უბრალოდ უმოკლესი გზის შეფასებას (shortest-path estimate). d და π მასივებს საწყისი მნიშვნელობები ვინიჭებათ შემდეგი პროცედურით, რომლის მუშაობის დროა $\Theta(V)$:

Algorithm 10: Initialize Single Source

Input: ორიენტირებული გრაფი $G = (V, E)$ და საწყისი წვერო s
Output: მიანიჭებს საწყისს მნიშვნელობებს d და π მასივებს

```
1 INITIALIZE-SINGLE-SOURCE(G, s) :
2 |  for ∀v ∈ V :
3 |  |  d[v] = ∞;
4 |  |  π[v] = NIL;
5 |  d[s] = 0;
```

$(u, v) \in E$ წიბოს რელაქსაცია შემდეგში მდგომარეობს: მოწმდება შეიძლება თუ არა v წვერომდე აქამდე არსებული უმოკლესი გზის გაუმჯობესება მისი u წვეროზე გატარებით? დადებითი პასუხის შემთხვევაში ხდება $d[v]$ და $\pi[v]$ ატრიბუტების განახლება. რელაქსაცია ამცირებს $d[v]$-ს $d[u] + w(u, v)$-მდე. ამავდროულად იცვლება $\pi[v]$-ც.

Algorithm 11: Relaxation

Input: ორიენტირებული გრაფი $G = (V, E)$, წონითი ფუნქცია $w : E \to R$, გრაფის წვეროები u და v
Output: ითვლის შემოწმების მომენტში უკეთესია თუ არა, რომ v წვეროში მოვხვდეთ u წვეროდან (u, v) წიბოს გამოყენებით

```
1 RELAX(u, v, w) :
2 |  if d[v] > d[u] + w(u,v) :
3 |  |  d[v] = d[u] + w(u,v);
4 |  |  π[v] = u;
```

ამ თავში აღწერილი ალგორითმებში ჯერ ხდება ინიციალიზაცია და შემდეგ რელაქსაცია. რელაქსაცია ერთადერთი პროცედურაა, რომელიც ცვლის $d[v]$ და $\pi[v]$ ატრიბუტებს. თუმცა ალგორითმები განსხვავდებიან იმით, თუ რამდენჯერ ტარდება რელაქსაცია და წიბოთა რა თანმიმდევრობისთვის. მაგ.: დეიქსტრას ალგორითმი აციკლური გრაფებისათვის მხოლოდ ერთხელ ახდენს წიბოთა რელაქსაციას, ხოლო ბელმან-ფორდის ალგორითმი - რამდენჯერმე.
განვიხილოთ უმოკლესი გზების და რელაქსაციას თვისებები, რომლებიც მოცემულია შემდეგ დებულებებში:

ლემა 4.2. (სამკუთხედის უტოლობა (triangle property)) ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით და $s \in V$ საწყისი წვეროთი. მაშინ ნებისმიერი $(u, v) \in E$-სათვის, გვაქვს:

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

Proof. ვთქვათ, არსებობს უმოკლესი p გზა s-დან v წვეროში, მაშინ ამ გზის წონა არ აღემატება ნებისმიერი სხვა გზის წონას s-დან v-ში, კერძოდ, ის არ აღემატება გზის წონას, რომელიც შედგება გზისგან s-დან u წვეროში და (u, v) წიბოსგან.
თუ უმოკლესი გზა s-დან v წვეროში არ არსებობს, მაშინ $\delta(s, v) = \infty$ ან $\delta(s, v) = -\infty$. $\delta(s, v) = \infty$ ნიშნავს, რომ წვერო არ არის მიღწევადი s-დან, მაგრამ მაშინ u წვეროც არ იქნება მიღწევადი $\delta(s, u) = \infty$ და დასამტკიცებელი უტოლობა სრულდება. თუ $\delta(s, v) = -\infty$, ეს ნიშნავს, რომ v წვერო არის უარყოფითწონიანი ციკლის წვერო და დასამტკიცებელი უტოლობა სრულდება. □

ლემა 4.3. (ზედა საზღვრის თვისება (upper-bound property)**)** ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით. ვთქვათ, $s \in V$ - საწყისი წვეროა. მაშინ INITIALIZE-SINGLE-SOURCE(G,s) პროცედურის შესრულებისა და წიბოთა ნებისმიერი თანმიმდევრობით რელაქსაციის შემდეგ, ნებისმიერი $v \in V$ წვეროსათვის სრულდება უტოლობა $d[v] \geq \delta(s, v)$. თუკი რომელიმე წვეროსათვის ეს უტოლობა გადაიქცევა ტოლობად, $d[v] = \delta(s, v)$ მაშინ იგი აღარ შეიცვლება.

Proof. დავამტკიცოთ $d[v] \geq \delta(s, v)$, ყოველი $v \in V$ წვეროსთვის ინდუქციის მეთოდის გამოყენებით, რელაქსაციის ბიჯების რაოდენობის მიმართ. $d[v] \geq \delta(s, v)$ უტოლობა სამართლიანია უშუალოდ ინიციალიზაციის შემდეგ, რადგან საწყისი წვეროსთვის, $d[s] = 0 \geq \delta(s, s)$ (შევნიშნოთ, რომ $\delta(s, s) = -\infty$, თუ s წვერო უარყოფითწონიანი ციკლის წვეროა; წინააღმდეგ შემთხვევაში, $\delta(s, s) = 0$), ხოლო $v \in V \backslash \{s\}$ წვეროებისთვის, ინიციალიზაციის შემდეგ, $d[v] = \infty$ და ამიტომ $d[v] \geq \delta(s, v)$.
ინდუქციის ბიჯად ჩავთვალოთ (u, v) წიბოს რელაქსაცია. ინდუქციის დაშვების თანახმად, ყველა $x \in V$ წვეროსთვის, რელაქსაციის წინ სრულდება უტოლობა $d[x] \geq \delta(s, x)$. დავამტკიცოთ, რომ უტოლობა სრულდება რელაქსაციის შემდეგაც. რელაქსაციის შემდეგ შეიძლება შეიცვალოს მხოლოდ $d[v]$, თუ ის შეიცვალა გვექნება:

$$d[v] = d[u] + w(u, v) \geq \delta(s, u) + w(u, v) \geq \delta(s, v)$$

სადაც პირველი უტოლობა ინდუქციის დაშვებიდან გამომდინარეობს, ხოლო მეორე - სამკუთხედის უტოლობიდან. ამრიგად, დამტკიცდა, რომ ნებისმიერი წვეროსათვის სრულდება უტოლობა $d[v] \geq \delta(s, v)$.
დავამტკიცოთ, რომ $d[v]$-ს მნიშვნელობა არ შეიცვლება მას შემდეგ, რაც შესრულდება $d[v] = \delta(s, v)$. რადგან $d[v] \geq \delta(s, v)$ სამართლიანია, $d[v]$ ვერ მიიღებს $\delta(s, v)$-ზე ნაკლებ მნიშვნელობას. $d[v]$-ს მნიშვნელობა ვერც გაიზრდება, რადგან რელაქსაციის შედეგად ის შეიძლება მხოლოდ შემცირდეს ან დარჩეს იგივე. □

ლემა 4.4. (არარსებული გზის თვისება (no-path property)**)**: ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი, წონითი $w : E \to R$ ფუნქციით და s საწყისი წვეროთი. ვთქვათ, $v \in V$ წვერო მიუღწევადია s-დან. მაშინ INITIALIZE-SINGLE-SOURCE(G,s) პროცედურის შესრულების შემდეგ გვაქვს $d[v] = \delta(s, v) = \infty$ და ეს ტოლობა არ შეიცვლება G გრაფში წიბოთა ნებისმიერი თანმიმდევრობით რელაქსაციის შემდეგაც.

Proof. ზედა საზღვრის თვისების თანახმად, სრულდება $\infty = \delta(s, v) \leq d[v]$, ამიტომ $d[v] = \infty = \delta(s, v)$. □

ლემა 4.5. ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით და ვთქვათ $(u, v) \in E$. მაშინ უშუალოდ ამ წიბოს რელაქსაციის შემდეგ სრულდება უტოლობა $d[v] \leq d[u] + w(u, v)$.

Proof. თუ უშუალოდ (u, v) წიბოს რელაქსაციამდე სრულდება $d[v] > d[u] + w(u, v)$, მაშინ ამ ოპერაციის შემდეგ, უტოლობა გადაიქცევა ტოლობად: $d[v] = d[u] + w(u, v)$, ხოლო თუ (u, v) წიბოს რელაქსაციამდე სრულდება $d[v] \leq d[u] + w(u, v)$, მაშინ რელაქსაციის პროცესში, არც $d[u]$, არც $d[v]$ არ შეიცვლება. ასე, რომ (u, v) წიბოს რელაქსაციის შემდეგ, $d[v] \leq d[u] + w(u, v)$. □

ლემა 4.6. (კრებადობის თვისება (convergence property)**)** ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი, წონითი $w : E \to R$ ფუნქციით და s საწყისი წვეროთი. ვთქვათ, $u, v \in V$ წვეროებისთვის არსებობს უმოკლესი გზა $s \sim u \to v$. ვთქვათ, შესრულდა INITIALIZE-SINGLE-SOURCE(G,s) პროცედურა, ხოლო შემდეგ წიბოთა გარკვეული თანმიმდევრობით რელაქსაცია, მათ შორის RELAX(u,v,w). თუ რაღაც მომენტში (u, v) წიბოს რელაქსაციამდე შესრულდა $d[u] = \delta(s, u)$ ტოლობა, მაშინ (u, v) წიბოს რელაქსაციის შემდეგ, ნებისმიერ მომენტში, შესრულდება ტოლობა $d[v] = \delta(s, v)$.

Proof. ზედა საზღვრის თვისების თანახმად, თუ რაღაც მომენტში (u, v) წიბოს რელაქსაციამდე შესრულდა $d[u] = \delta(s, u)$ ტოლობა, ის შემდგომაც არ შეიცვლება. კერძოდ, (u, v) წიბოს რელაქსაციის შემდეგ. მივიღებთ:

$$d[v] \leq d[u] + w(u, v) = \delta(s, u) + w(u, v) = \delta(s, v)$$

სადაც უტოლობა გამომდინარეობს **ლემა** 4.5-დან, ხოლო ბოლო ტოლობა **ლემა** 4.1-დან. ზედა საზღვრის თვისების თანახმად, $d[v] \geq \delta(s, v)$. ამიტომ, შეიძლება დავასკვნათ, რომ $d[v] = \delta(s, v)$ და ის აღარ შეიცვლება. □

ლემა 4.7. (გზის რელაქსაციის თვისება (path-relaxation property)) ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით და s საწყისი წვეროთი. განვიხილოთ ნებისმიერი უმოკლესი გზა $p =< v_0, v_1, \ldots, v_k > s = v_0$ წვეროდან v_k წვერომდე. ვთქვათ, შესრულდა INITIALIZE-SINGLE-SOURCE(G,s) პროცედურა, ხოლო შემდეგ, წიბოების შემდეგი თანმიმდევრობით რელაქსაცია:
$(v_0, v_1), (v_1, v_2), \ldots, (v_{k-1}, v_k)$, მაშინ ამ რელაქსაციების შემდეგ და, შემდგომაც, ნებისმიერ მომენტში, სრულდება ტოლობა $d[v_k]\delta(s, v_k)$. ეს თვისება სამართლიანია, მიუხედავად იმისა, ხდება თუ არა რელაქსაცია სხვა წიბოებზე, მათ შორის რელაქსაცია იმ წიბოებზე, რომლებიც ენაცვლება p გზის წიბოებს.

Proof. ინდუქციით დავამტკიცოთ, რომ p გზის i-ური წიბოს რელაქსაციის შემდეგ, სრულდება: $d[v_i] = \delta(s, v_i)$. ბაზისად ავიღოთ $i = 0$. მანამ, სანამ p გზაში შემავალი ერთი მაინც წიბოს რელაქსაცია მოხდება, ინიციალიზაციის შემდეგ, $d[v_0] = d[s] = 0 = \delta(s, s)$. ზედა საზღვრის თვისების თანახმად, $d[s]$-ს მნიშვნელობა აღარ შეიცვლება.
ვთქვათ, $(i-1)$-ური წიბოს რელაქსაციის შემდეგ სრულდება: $d[v_{i-1}] = \delta(s, v_{i-1})$. განვიხილოთ i-ური, (v_{i-1}, v_i) წიბოს რელაქსაცია. კრებადობის თვისების თანახმად, ამ წიბოს რელაქსაციის შედეგად, $d[v_i] = \delta(s, v_i)$ და ეს ტოლობა აღარ შეიცვლება. □

ლემა 4.8. (წინამორბედობის ქვეგრაფის თვისება (predecessor subgraph property)) ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი, წონითი $w : E \to R$ ფუნქციით და s საწყისი წვეროთი. ვთქვათ, G გრაფი არ შეიცავს s წვეროდან მიღწევად უარყოფითწონიან ციკლებს. ვთქვათ, შესრულდა INITIALIZE-SINGLE-SOURCE(G,s) პროცედურა, ხოლო შემდეგ, რაიმე თანმიმდევრობით G გრაფის წიბოების რელაქსაცია, რომლის შედეგადაც ყოველი $v \in V$ წვეროსთვის სრულდება ტოლობა $d[v] = \delta(s, v)$, მაშინ წინამორბედობის ქვეგრაფი G_π წარმოადგენს s ფესვის მქონე უმოკლესი გზების ხეს.

4.2 ბელმან-ფორდის ალგორითმი

ბელმან-ფორდის ალგორითმი (Bellman-Ford algorithm) ხსნის საწყისი წვეროდან უმოკლესი გზების პოვნის ამოცანას ზოგად შემთხვევაში, როცა ნებისმიერ წიბოს შესაძლოა ჰქონდეს უარყოფითი წონა. ამ ალგორითმის ღირსებად შეიძლება ჩაითვალოს ისიც, რომ ის განსაზღვრავს არსებობს თუ არა გრაფში საწყისი წვეროდან მიღწევადი უარყოფითწონიანი ციკლი. ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით და s საწყისი წვეროთი. ბელმან-ფორდის ალგორითმი იძლევა TRUE მნიშვნელობას, თუ გრაფში საწყისი წვეროდან არაა მიღწევადი უარყოფითწონიანი ციკლი და იძლევა მნიშვნელობას FALSE, თუკი ასეთი ციკლი საწყისი წვეროდან მიღწევადია. პირველ შემთხვევაში ალგორითმი პოულობს უმოკლეს გზებს და მათ წონებს, ხოლო მეორე შემთხვევაში - უმოკლესი გზა არ არსებობს.

Algorithm 12: Bellman-Ford (Single Source Shortest Paths)

Input: ორიენტირებული გრაფი $G = (V, E)$, წონითი ფუნქცია $w : E \to R$ და საწყისი წვერო s
Output: d-ში გამოითვლის უმოკლეს მანძილებს s-დან ყველა სხვა წვერომდე, π-ში გამოითვლის ხეს, ალგორითმი დააბრუნებს TRUE-ს თუ გრაფში არ არსებობს უარყოფითი წონის ციკლი, წინააღმდეგ შემთხვევაში FALSE-ს

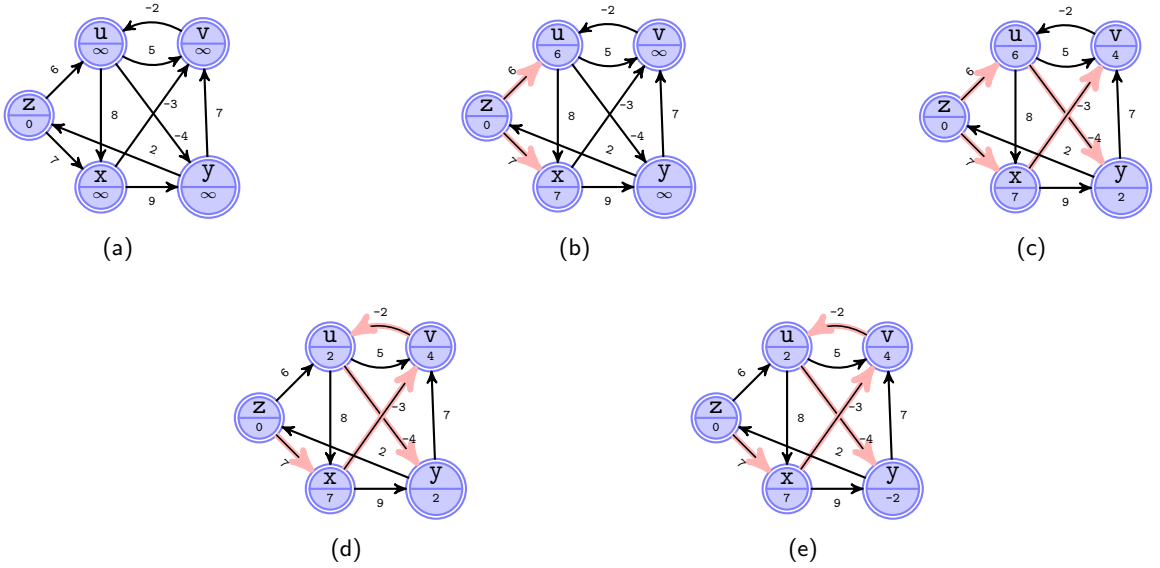
```
1 BELLMAN-FORD(G, w, s) :
2 |  INITIALIZE-SINGLE-SOURCE(G, s);
3 |  for i=1; i<|V|; i++ :
4 |  |  for ∀(u, v) ∈ E :
5 |  |  |  RELAX(u, v, w);
6 |  for ∀(u, v) ∈ E :
7 |  |  if d[v] > d[u] + w(u,v) :
8 |  |  |  return FALSE;
9 |  return TRUE;
```

ვნახოთ, როგორ მუშაობს ალგორითმი. სტრ. 2-ში ხდება ინიციალიზაცია. შემდეგ ალგორითმი $(|V| - 1)$-ჯერ იმეორებს ერთსა და იმავე მოქმედებას: ახდენს გრაფის თითოეული წიბოს რელაქსაციას (3-5 სტრიქონები). შემდეგ, ალგორითმი ამოწმებს, არსებობს თუ არა საწყისი წვეროდან მიღწევადი უარყოფითწონიანი ციკლი (6-8 სტრიქონები) და აბრუნებს შესაბამის მნიშვნელობას.
სურ. 4.3-ზე მოცემულია ბელმან-ფორდის ალგორითმის მუშაობის პროცესი. საწყისი წვეროა z. წვეროებში ნაჩვენებია უმოკლესი გზების შეფასებები (ატრიბუტი d), ხოლო გამოყოფილი წიბოები მიუთითებენ მშობლების

მნიშვნელობებს: თუ გამოყოფილია (u, v) წიბო, $\pi(v) = u$. ამასთან, წვეროების დამუშავება ხდება ლექსიკოგრაფიულად დალაგებული შემდეგი თანმიმდევრობით:

$$(u, v)(u, x)(u, y)(v, u)(x, v)(x, y)(y, v)(y, z)(z, u)(z, x)$$

ა)-ზე ნაჩვენებია ინიციალიზაციის პროცედურის შემდეგ, უშუალოდ წიბოების რელაქსაციის წინ არსებული სიტუაცია, ბ)-ც)-დ)-ზე ნაჩვენებია წიბოების რელაქსაციის შედეგად მიღებული მდგომარეობა, ხოლო ე)-ზე - საბოლოო მდგომარეობა.



ნახ. 4.3:

ბელმან-ფორდის ალგორითმის მუშაობის დროა $O(VE)$. ინიციალიზაციას სჭირდება $\Theta(V)$ დრო; 3-5 სტრიქონებში, თითოეულ ჯერზე წიბოების რელაქსაციას - $\Theta(E)$ დრო (სულ $(|V|-1)$-ჯერ); 6-8 სტრიქონებში ციკლის შესრულებას კი - $O(E)$.
ქვემოთ მოყვანილი თეორემა და მისი შედეგი, ამტკიცებს ბელმან-ფორდის ალგორითმის კორექტულობას.

ლემა 4.9. ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით და s საწყისი წვეროთი, რომელიც არ შეიცავს s წვეროდან მიღწევად უარყოფითწონიან ციკლს, მაშინ ბელმან-ფორდის ალგორითმის 3-5 სტრიქონებში, for ციკლის $(|V| - 1)$ იტერაციის დასრულების შემდეგ, ყოველი $v \in V$ s-დან მიღწევადი წვეროსთვის სრულდება $d[v] = \delta(s, v)$.

Proof. განვიხილოთ რაიმე s-დან მიღწევადი წვერო $v \in V$. ვთქვათ, $p =< v_0, v_1, \ldots, v_k >$, $v_0 = s$ $v_k = v$ უმოკლესი აციკლური გზაა s-დან v-ში. p გზა შეიცავს არაუმეტეს $(|V|-1)$ წიბოს, რაც ნიშნავს, რომ $k \leq |V|-1$. for ციკლის (3-5 სტრ.) ყოველი $(|V|-1)$ იტერაციის დროს, ხდება ყველა $|E|$ წიბოს რელაქსაცია, i-ური $(i = 1, 2, \ldots, k)$ იტერაციის დროს რელაქსირებულ წიბოებს შორის არის წიბო (v_{i-1}, v_i). ამიტომ, **ლემა** 4.7-ს თანახმად, სრულდება: $d[v] = d[v_k] = \delta(s, v_k) = \delta(s, v)$. □

შედეგი 4.1. ვთქვათ, მოცემული გვაქვს ორიენტირებული წონადი $G = (V, E)$ გრაფი წონითი $w : E \to R$ ფუნქციით და s საწყისი წვეროთი. მაშინ, ყოველი $v \in V$ წვეროსთვის, გზა s-დან v-ში არსებობს მაშინ და მხოლოდ მაშინ, როცა G გრაფის ბელმან-ფორდის ალგორითმით დამუშავების შემდეგ, სრულდება: $d[v] < \infty$.

თეორემა 4.1. (ბელმან-ფორდის ალგორითმის კორექტულობა) ვთქვათ, ბელმან-ფორდის ალგორითმით ხდება s საწყისი წვეროს და $w : E \to R$ წონითი ფუნქციის მქონე ორიენტირებული, წონადი $G = (V, E)$ გრაფის დამუშავება. თუ G გრაფი არ შეიცავს s წვეროდან მიღწევად უარყოფითწონიან ციკლებს, მაშინ ალგორითმი აბრუნებს TRUE მნიშვნელობას, ყოველი $v \in V$ წვეროსთვის, სრულდება $d[v] = \delta(s, v)$ და წინამორბედობის ქვეგრაფი G_π არის s ფესვის მქონე უმოკლესი გზების ხე. ხოლო თუ G გრაფი შეიცავს s წვეროდან მიღწევად უარყოფითწონიან ციკლს, მაშინ ალგორითმი აბრუნებს FALSE მნიშვნელობას.

Proof. დავუშვათ, G გრაფი არ შეიცავს s წვეროდან მიღწევად უარყოფითწონიან ციკლს. ჯერ დავამტკიცოთ, რომ ალგორითმის მუშაობის დასრულების შემდეგ, ყოველი $v \in V$ წვეროსთვის, სრულდება $d[v] = \delta(s, v)$. თუ v წვერო მიღწევადია s-დან, ეს ტოლობა გამომდინარეობს **ლემა** 4.9-დან, ხოლო თუ v არ არის მიღწევადი s-დან - არარსებული გზის თვისებიდან. ამ დებულებიდან და წინამორბედობის ქვეგრაფის თვისებიდან (**ლემა** 4.8)

გამომდინარეობს, რომ G_π გრაფი არის უმოკლესი გზების ხე. ახლა, ვაჩვენოთ, რომ ბელმან-ფორდის ალგორითმი აბრუნებს TRUE მნიშვნელობას. ალგორითმის დასრულების შემდეგ, ყოველი $(u,v) \in E$ წიბოსთვის სრულდება:

$$d[v] = \ delta(s,v) \leq \delta(s,u) + w(u,v) = d[u] + w(u,v)$$

ამიტომ სტრ. 7-ში, არც ერთი შედარების შედეგად ალგორითმი არ დააბრუნებს FALSE მნიშვნელობას.
ახლა, განვიხილოთ შემთხვევა, როცა G გრაფი შეიცავს s წვეროდან მიღწევად უარყოფითწონიან ციკლს. ვთქვათ, ეს ციკლია $c =< v_0, v_1, \ldots, v_k >$, სადაც $v_0 = v_k$, მაშინ:

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0 \tag{4.1}$$

დავუშვათ საწინააღმდეგო, ვთქვათ, ალგორითმი აბრუნებს TRUE მნიშვნელობას. (რაც, აგრეთვე, ნიშნავს, რომ ალგორითმი აბრუნებს უმოკლეს გზებს და მათ წონებს) მაშინ ყოველი $(i = 1, 2, \ldots, k)$-სთვის სრულდება: $d[v_i] \leq d[v_{i-1}] + w(v_{i-1}, v_i)$. განვიხილოთ ამ უტოლობის ჯამი ციკლის ყველა წვეროსთვის:

$$\sum_{i=1}^{k} d[v_i] \leq \sum_{i=1}^{k} (d[v_{i-1}] + w(v_{i-1}, v_i)) = \sum_{i=1}^{k} d[v_{i-1}] + \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

რადგანაც, $v_0 = v_k$, ამიტომ c ციკლის ყოველი წვერო $\sum_{i=1}^{k} d[v_i]$ და $\sum_{i=1}^{k} d[v_{i-1}]$ ჯამებში გვხვდება მხოლოდ ერთხელ და $\sum_{i=1}^{k} d[v_i] = \sum_{i=1}^{k} d[v_{i-1}]$. **შედეგი** 4.10-ის თანახმად $d[v_i]$ ატრიბუტი იღებს სასრულ მნიშვნელობებს, ამრიგად სამართლიანია უტოლობა $\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$, რაც ეწინააღმდეგება (4.1)-ს. მივიღეთ წინააღმდეგობა. ამრიგად, ბელმან-ფორდის ალგორითმი აბრუნებს TRUE მნიშვნელობას, თუ გრაფი არ შეიცავს საწყისი წვეროდან მიღწევად უარყოფითწონიან ციკლს და აბრუნებს FALSE მნიშვნელობას, წინააღმდეგ შემთხვევაში. □

4.3 უმოკლესი გზები აციკლურ ორიენტირებულ გრაფში

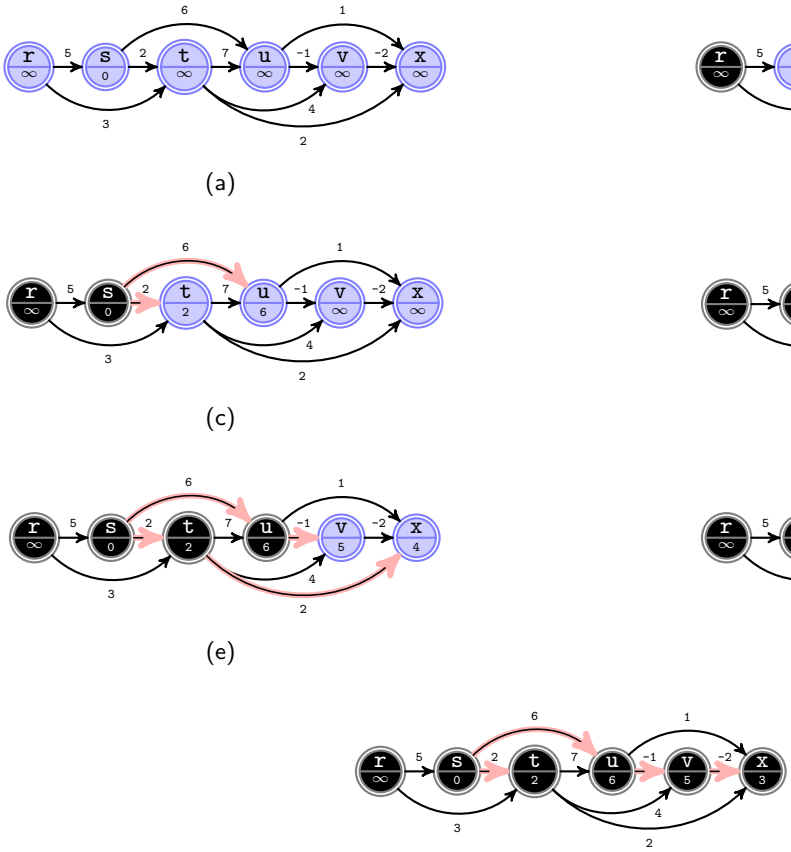
აციკლურ ორიენტირებულ $G = (V, E)$ გრაფში ერთი წვეროდან უმოკლესი გზების მოსაძებნად საჭიროა $\Theta(V+E)$ დრო, თუ წიბოების რელაქსაციას ჩავატარებთ ტოპოლოგიურად სორტირებული წვეროების მიხედვით. შევნიშნოთ, რომ აციკლურ ორიენტირებულ გრაფში უმოკლესი გზები ყოველთვის განსაზღვრულია, რადგან ციკლები (მათ შორის, უარყოფითწონიანიც) საერთოდ არ გვხვდება.
ტოპოლოგიური სორტირება იმგვარად განალაგებს წვეროებს წრფივი მიმდევრობით, რომ ყველა წიბო ერთნაირი მიმართულებისაა. ამის შემდეგ უნდა განვიხილოთ წვეროები ამ მიმდევრობით (წიბოს დასაწყისი ყოველთვის მის ბოლოზე ადრე იქნება განხილული) და ყოველი წვეროსათვის მოვახდინოთ მისგან გამომავალი ყველა წიბოს რელაქსაცია.

Algorithm 13: DAG Shortest Paths (Single Source Shortest Paths)

Input: აციკლური ორიენტირებული გრაფი $G = (V, E)$, წონითი ფუნქცია $w : E \to R$ და საწყისი წვერო s
Output: d-ში გამოითვლის უმოკლეს მანძილებს s-დან ყველა სხვა წვერომდე, π-ში გამოითვლის ხეს

```
1 DAG-SHORTEST-PATHS(G, w, s) :
2 |  L = TOPOLOGICAL-SORT(G);
3 |  INITIALIZE-SINGLE-SOURCE(G, s);
4 |  for ∀u ∈ L :
5 |  |  for ∀v ∈ Adj[u] :
6 |  |  |  RELAX(u, v, w);
7 |  return d, π;
```

ალგორითმის მუშაობის პროცესი ნაჩვენებია სურ. 4.4-ზე. საწყისი წვეროა s. ტოპოლოგიურ სორტირებაზე (სტრ. 2) იხარჯება $\Theta(V + E)$ დრო, ხოლო ინიციალიზაციაზე (მე-3 სტრ.) - $\Theta(V)$. წვეროსთვის ციკლში (4-6 სტრ.) სრულდება ერთი ოპერაცია, ყოველი წვეროდან გამომავალი წიბოები ერთხელ დამუშავდება, ამრიგად, შიდა ციკლში სულ სრულდება $|E|$ იტერაცია (5-6 სტრ.) თითოეული იტერაციის ღირებულება $\Theta(1)$-ია. ალგორითმის მუშაობის დროა $\Theta(V + E)$ და ის გამოისახება მოსაზღვრე წვეროთა სიის ზომის წრფივი ფუნქციით.



ნახ. 4.4:

აღწერილი ალგორითმი შესაძლებელია გამოვიყენოთ ე.წ. "კრიტიკული გზების" საპოვნელად. განვიხილოთ ამოცანა, სადაც ორიენტირებული, აციკლური გრაფის ყოველი წიბო წარმოადგენს რაღაც საქმიანობას, ხოლო წიბოს წონა - მის შესასრულებლად საჭირო დროს. თუკი გვაქვს წიბოები (u, v) და (v, x), მაშინ (u, v) წიბოს შესაბამისი სამუშაო უნდა შესრულდეს (v, x) წიბოს შესაბამისი სამუშაოს დაწყებამდე. **კრიტიკული გზა** (critical path) - ესაა უგრძელესი გზა გრაფში, რომლის წონა ტოლია ყველა სამუშაოს შესასრულებლად დახარჯული დროსი, თუკი მაქსიმალურადაა გამოყენებული ზოგიერთი სამუშაოს პარალელურად შესრულების შესაძლებლობა. კრიტიკული გზის წონა არის ყველა სამუშაოს შესრულების სრული დროის ქვედა შეფასება. კრიტიკული გზის საპოვნელად ყველა წონის ნიშანი უნდა შეიცვალოს საპირისპიროთი და შესრულდეს DAG-SHORTEST-PATHS ალგორითმი.

4.4 დეიქსტრას ალგორითმი

დეიქსტრას ალგორითმი პოულობს $G = (V, E)$ ორიენტირებული გრაფისათვის უმოკლეს გზებს საწყისი s წვეროდან ყველა დანარჩენ წვერომდე. აუცილებელია, რომ ყველა წიბოს წონა იყოს არაუარყოფითი $w(u, v) \geq 0$ ყოველი $(u, v) \in E$.

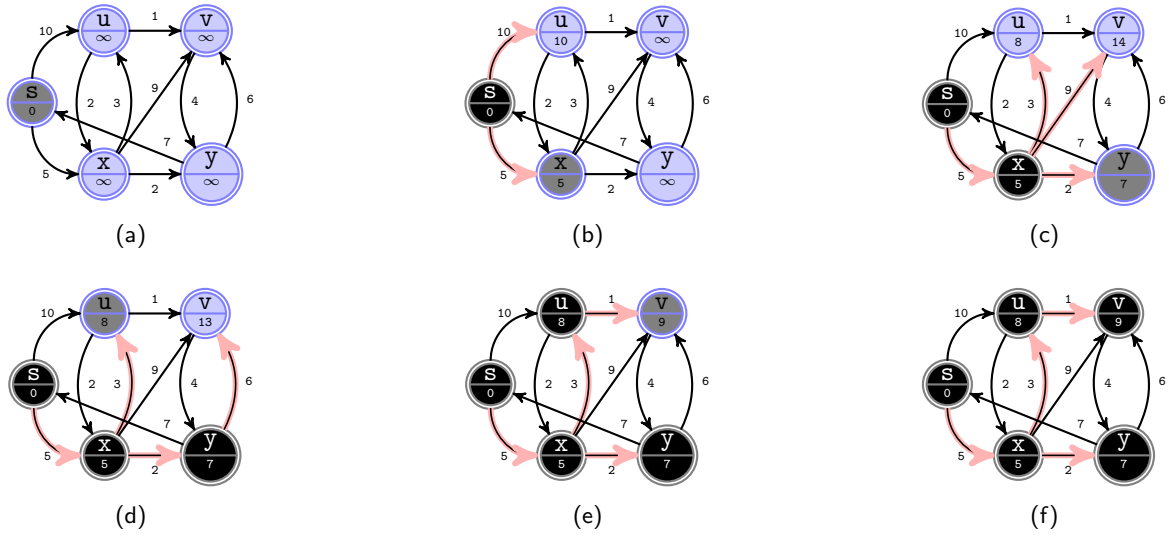
დეიქსტრას ალგორითმის მუშაობის დროს გამოიყენება $S \subseteq V$ სიმრავლე, რომელიც შედგება იმ v წვეროებისაგან, რომელთათვისაც $\delta(s, v)$ უკვე მოძებნილია (ე.ი. $d[v] = \delta(s, v)$). ალგორითმი ირჩევს უმცირესი $d[u]$-ს მქონე $u \in V \backslash S$ წვეროს, ამატებს u-ს S სიმრავლეში და ახდენს u-დან გამომავალი ყველა წიბოს რელაქსაციას, რის შემდეგაც ციკლი მეორდება. წვეროები, რომლებიც S-ს არ მიეკუთვნებიან, ინახება Q პრიორიტეტებიან რიგში, რომლის გასაღებიც განისაზღვრება d ფუნქციის მნიშვნელობებით. იგულისხმება, რომ გრაფი მოცემულია მოსაზღვრე

წვეროთა სიით.

Algorithm 14: Dijkstra Shortest Paths

Input: აციკლური ორიენტირებული გრაფი $G = (V, E)$, წონითი ფუნქცია $w : E \to R^+$ და საწყისი წვერო s
Output: d-ში გამოითვლის უმოკლეს მანძილებს s-დან ყველა სხვა წვერომდე, π-ში გამოითვლის ხეს

```
1  DIJKSTRA(G, w, s) :
2      INITIALIZE-SINGLE-SOURCE(G, s);
3      S = ∅;
4      Q = V; // პრიორიტეტული რიგი, დ გვაძლევს პრიორიტეტს
5      while Q ≠ ∅ :
6          u = EXTRACT-MIN(Q);
7          S = S ∪ {u};
8          for ∀v ∈ adj[u] :
9              RELAX(u, v, w);
10     return d, π;
```



ნახ. 4.5:

დეიქსტრას ალგორითმის მუშაობის პროცესი აღწერილია სურ. 4.5-ზე. საწყისი წერტილია s. წვეროებში ჩაწერილია უმოკლესი გზების შეფასებები მოცემული მომენტისათვის. შავი ფერით აღნიშნულია წვეროები, რომლებიც S სიმრავლეს ეკუთვნიან. სხვა წვეროები დგანან $Q = V \backslash S$ პრიორიტეტებიან რიგში. რუხი ფერის წვეროები ციკლის მომდევნო იტერაციის დროს გამოდიან u წვეროს როლში. d და π თავიანთ საბოლოო მნიშვნელობებს ღებულობენ სურ. 4.5ფ-ზე.
ალგორითმის მუშაობის სტრ. 2-ში ხდება d-ს და π-ს, მე-3 სტრიქონში - S-ის, ხოლო მე-4 სტრიქონში - Q-ს ინიციალიზაცია. while ციკლის ყოველი იტერაციის წინ, 5-9 სტრ-ში $Q = V \backslash S$. დასაწყისში $Q = V$. 5-9 სტრიქონებში while ციკლის ყოველი იტერაციის დროს Q-დან ხდება უმცირესი $d[u]$-ს მქონე u წვეროს ამოღება და ის ემატება S სიმრავლეს (თავდაპირველად $u = s$). 8-9 სტრიქონებში ხდება u-დან გამოსული ყოველი (u, v) წიბოს რელაქსაცია. ამ დროს შეიძლება შეიცვალოს $d[v]$ შეფასება და $\pi[v]$ მშობელი. შევნიშნოთ, რომ ციკლის მუშაობის დროს Q რიგში ახალი წვეროები არ ემატება, ხოლო Q-დან ამოღებული ყოველი წვერო ემატება S სიმრავლეს მხოლოდ ერთხელ, ამიტომ while ციკლის იტერაციათა რაოდენობაა $|V|$.
რადგან დეიქსტრას ალგორითმში S სიმრავლეს ემატება $V \backslash S$ სიმრავლიდან ამოღებული ყველაზე "მსუბუქი" წვერო, ამიტომ ამბობენ, რომ ალგორითმი მუშაობს ხარბი სტრატეგიით, რომელიც ყოველთვის არ იძლევა ოპტიმალურ შედეგს. ქვემომოყვანილი თეორემა და მისი შედეგი, რომელიც დაუმტკიცებლად მოგვყავს, ამტკიცებს დეიქსტრას ალგორითმის კორექტულობას.

თეორემა 4.2. (დეიქსტრას ალგორითმის კორექტულობა) s საწყისი წვეროს და წონითი $w : E \to R$ ფუნქციის მქონე ორიენტირებული, წონადი $G = (V, E)$ გრაფის, დეიქსტრას ალგორითმით დამუშავების შემდეგ, ყოველი $u \in V$-სათვის სრულდება $d[u] = \delta(s, u)$.

შედეგი 4.2. s საწყისი წვეროს და წონითი $w : E \to R$ ფუნქციის მქონე ორიენტირებული, წონადი $G = (V, E)$

გრაფის დეიქსტრას ალგორითმით დამუშავების შემდეგ, წინამორბედობის ქვეგრაფი G_π წარმოადგენს უმოკლესი გზების ხეს, რომლის ფესვიც არის s წვერო.

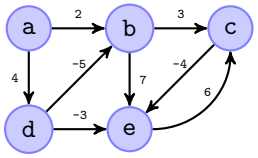
დეიქსტრას ალგორითმის მუშაობის დრო. ამ ალგორითმში გამოიყენება პრიორიტეტებიანი Q რიგი და სამი ოპერაცია (INSERT (სტრ. 4), EXTRACT-MIN (სტრ. 6), DECREASE-KEY (არაცხადად მონაწილეობს RELAX-ში) სტრ. 9) INSERT და EXTRACT-MIN პროცედურების გამოძახება ხდება ყოველი წვეროსთვის ერთხელ (წვეროების რაოდენობაა $|V|$). რადგან, ყოველი წვერო S სიმრავლეში ემატება ერთხელ, ალგორითმის მუშაობის პროცესში, ყოველი წიბოს (რომელთა რაოდენობაა $|E|$) დამუშავება მოსაზღვრე წვეროთა სიაში ხდება ერთხელ (სტრ. 8-9) ე.ი. სრულდება DECREASE-KEY-ს არა უმეტეს $|E|$ ოპერაციისა.
თუ პრიორიტეტებიანი Q რიგი რეალიზებულია როგორც მასივი, მაშინ EXTRACT-MIN ოპერაციას დასჭირდება $O(V)$ დრო; ალგორითმი ასრულებს ამ ოპერაციას $|V|$-ჯერ, ამიტომ რიგიდან ყველა ელემენტის ამოღებას დასჭირდება $O(V^2)$ დრო. ყველა დანარჩენ ოპერაციას სჭირდება $O(E)$ დრო. INSERT და DECREASE-KEY პროცედურებს სჭირდებათ $O(1)$ დრო. ალგორითმის მუშაობის დროა $O(V^2 + E^2) = O(V^2)$
თუ გრაფი ხალვათია ($|E| \ll |V|^2$), აზრი აქვს Q რიგის რეალიზებას, ორობითი გროვის საშუალებით. ორობითი გროვის აგებას დასჭირდება $O(V)$ დრო, EXTRACT-MIN ოპერაციას დასჭირდება $O(\log V)$ დრო, ასეთი ოპერაციების რაოდენობაა $|V|$, DECREASE-KEY ოპერაციას დასჭირდება $O(\log V)$ დრო, ასეთი ოპერაციების რაოდენობაა არა უმეტეს $\|$; ხოლო დეიქსტრას ალგორითმის მუშაობის სრული დრო იქნება $O((V+E)\log V) = O(E \log V)$, თუ ყველა წვერო მიღწევადია საწყისი წვეროდან. თუ პრიორიტეტებიანი Q რიგი რეალიზებულია ფიბონაჩის გროვის სახით, ალგორითმის მუშაობის დრო შეიძლება შემცირდეს $O(V \log V + E)$-მდე.

4.5 იენის ალგორითმი

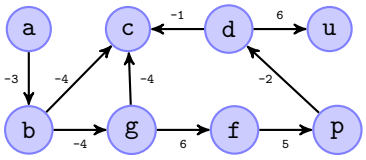
(ბელმან-ფორდის ალგორითმის მოდიფიკაცია) G გრაფის წვეროები გადავნომროთ ნებისმიერად და გრაფის წიბოთა E სიმრავლე გავყოთ ორ ნაწილად: E_f - წიბოები, რომლებიც მიმართულია ნაკლები ნომრის მქონე წვეროდან მეტი ნომრის მქონე წვეროსაკენ და E_b - წიბოები, რომლებიც მიმართულია მეტი ნომრის მქონე წვეროდან ნაკლები ნომრის მქონე წვეროსაკენ. ვთქვათ, $G_f = (V, E_f)$ და $G_b = (V, E_b)$, სადაც V გრაფის წვეროთა სიმრავლეა. ცხადია, რომ G_f და G_b გრაფები აციკლურია და ორივე მათგანი დალაგებულია ტოპოლოგიურად.
ბელმან-ფორდის ალგორითმის ციკლის ყოველ იტერაციაზე მოვახდინოთ წიბოთა რელაქსაცია შემდეგნაირად: ჯერ გადავარჩიოთ წვეროები ნომრების ზრდადობის მიხედვით და ყოველი წვეროსათვის მოხდეს მისგან გამომავალი E_f გრაფის ყველა წიბოს რელაქსაცია, შემდეგ წვეროები ნომრების კლებადობის მიხედვით და ყოველი წვეროსათვის მოხდეს მისგან გამომავალი E_b გრაფის ყველა წიბოს რელაქსაცია. ციკლის $|V|/2$ იტერაციის შემდეგ გრაფის ყველა წვეროსათვის შესრულებული იქნება $d[v] = \delta(s, v)$.

4.6 სავარჯიშოები

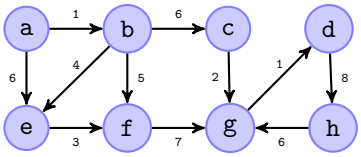
1. ბელმან-ფორდის ალგორითმით იპოვეთ უმოკლესი გზები a წვეროდან შემდეგ გრაფში:



2. შემდეგ გრაფში შეასრულეთ DAG-SHORTEST-PATHS(G,w,a):



3. დეიქსტრას ალგორითმით იპოვეთ უმოკლესი გზები a წვეროდან შემდეგ გრაფში:



4. მოიყვანეთ უარყოფითი წონის მქონე წიბოს შემცველი ორიენტირებული გრაფის მაგალითი, რომლისთვისაც დეიქსტრას ალგორითმი იძლევა არასწორ შედეგს.