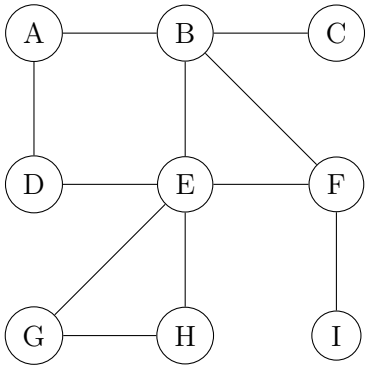
Sedme vaje APS2: sprehodi po grafih

Prerezne točke

Naloga

Vozlišče v povezanem neusmerjenem grafu je *prerezna točka*, če njegova odstranitev povzroči razpad grafa na najmanj dva dela. Na primer, v grafu na sliki 1 so prerezne točke vozlišča B, E in F.[1]



Slika 1: Primer povezanega neusmerjenega grafa.

Prerezne točke zlahka poiščemo v času $O(VE)$: vsako vozlišče posebej odstranimo iz grafa in s pomočjo iskanja v širino (BFS) ali iskanja v globino (DFS) preverimo, ali je graf še vedno povezan. Če nekoliko razmislimo, pa se izkaže, da lahko takšna vozlišča poiščemo tudi v času $O(E)$.

DFS

Algoritem za iskanje prereznih točk temelji na razširjenem algoritmu preiskovanja v globino (DFS), ki smo ga spoznali na predavanjih, a ne bo nič narobe, če ga ponovimo. Če DFS uporabljamo na povezanem neusmerjenem grafu, zadošča, da njegovo rekurzivno jedro poženemo samo enkrat, saj bo odkril vsa vozlišča, in to ne glede na to, v katerem vozlišču začnemo. Dogovorimo se, da začnemo v vozlišču 0 (torej A), sosede pa obravnavamo po naraščajočih indeksih (torej po abecedi). Algoritem je prikazan kot algoritem 1.

Algoritem med izvajanjem vzdržuje štiri atribute vozlišč:

- *Barva* vozlišča je lahko bela, siva ali črna. Bela barva predstavlja neobiskano vozlišče, siva predstavlja vozlišče, ki smo ga že obiskali, nismo pa še preiskali vseh njegovih sosedov, črna barva pa predstavlja vozlišče, pri katerem smo preiskali že vse njegove sosede. Na začetku (pred izvedbo algoritma) so vsa vozlišča bela.
- Če pri izvajanju v vozlišču u odkrijemo neobiskanega soseda v, potem u postane *predhodnik* vozlišča v v DFS-gozdu. DFS-gozd je v primeru povezanega neusmerjenega grafa kar drevo.
- *Čas odkritja* vozlišča ($v.d$) je trenutek, ko DFS poženemo na tem vozlišču, torej trenutek, ko vozlišče postane sivo.
- *Čas zaključka* vozlišča ($v.f$) je trenutek, ko DFS zaključimo s pregledom vseh sosedov vozlišča, torej trenutek, ko vozlišče postane črno.

[1] A = 0, B = 1, C = 2 itd. Črkovne oznake uporabljamo zgolj zaradi lažjega sklicevanja. Ta razlog bo postal še bolj smiseln, ko bomo dopisali številske atribute.

Algoritem 1 Iskanje v globino (DFS) za povezane neusmerjene grafe.

```
function DFS(u)
    u.barva ← siva
    čas ← čas + 1
    u.d ← čas
    for all v ∈ Adj[u] do                          ▷ po tabeli sosedov vozlišča u
        if v.barva = bela then
            ▷ (u, v) je drevesna povezava                                    ◁
            v.predhodnik ← u
            DFS(v)
        else if v ≠ u.predhodnik then
            ▷ (u, v) je povratna povezava                                    ◁
    čas ← čas + 1
    u.f ← čas
    u.barva ← črna

▷ V glavnem delu programa, npr. funkciji main:                               ◁
for v ← 0 to n − 1 do
    v.barva ← bela
    v.predhodnik ← −1
    v.d ← −1
    v.f ← −1
čas ← 0
DFS(0)
```

Časa ne merimo v minutah in sekundah, pač pa gre preprosto za globalni števec, ki ga inicializiramo na 0, ob vsakem dogodku pa ga povečamo za 1.

Algoritem DFS poleg vozlišč preišče tudi povezave in jih razdeli v dve skupini. V prvi skupini so povezave, ki tvorijo DFS-drevo. To so *drevesne povezave*; odkrijemo jih, ko pri izvajanju na določenem vozlišču naletimo na belega (tj. neodkritega) soseda. *Povratna povezava* pa je povezava med vozliščem in njegovim prednikom v DFS-drevesu. Odkrijemo jo, ko pri izvajanju na vozlišču u naletimo na sivega (tj. že odkritega) soseda v, ki pa ni neposredni predhodnik vozlišča u (sicer bi vsaka drevesna povezava bila tudi povratna). Pri usmerjenih grafih imamo še tretjo možnost (povezave med trenutnim in črnim vozliščem), pri neusmerjenih pa takih povezav ni.

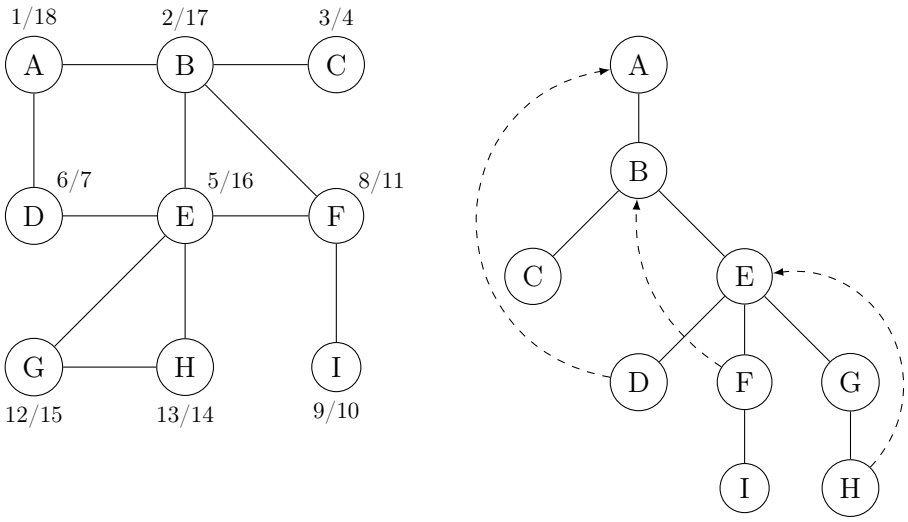
Za lažje razumevanje smo algoritem DFS pognali na grafu s slike 1. Slika 2 prikazuje čase odkritja in zaključka za posamezna vozlišča ter DFS-drevo. Črtkane črte v DFS-drevesu predstavljajo povratne povezave. Te pripadajo izhodiščnemu grafu, ne pa tudi DFS-drevesu.

Zapišimo še trditev, ki smo jo dokazali na predavanjih (oziroma je (še) nismo, čeprav bi jo (že) morali):

Trditev 1 (Izrek o beli poti)**.** *Vozlišče v postane potomec vozlišča u v DFS-drevesu natanko v primeru, če je pot od vozlišča u do vozlišča v v trenutku odkritja vozlišča u sestavljena iz samih belih vozlišč (vključno z vozliščem v).*

Póti, sestavljeni iz samih belih vozlišč, bomo rekli enostavno *bela pot*. V nasprotju s sivo potjo me ne bo vodila na Gorenjsko, mi bo pa pomagala pri dokazovanju nadaljnjih trditev.

Definirajmo še nekaj izrazov, povezanih z DFS-drevesom. Saj jih že poznamo, a osvežitev ne bo odveč:



Slika 2: Časi odkritja in zaključka posameznih vozlišč ter DFS-drevo.

- *Predhodnik* vozlišča pomeni isto kot *starš* vozlišča.
- *Naslednik* vozlišča pomeni isto kot *otrok* vozlišča.
- *Potomci* vozlišča u so vozlišče u sámo ter njegovi otroci, vnuki itd.
- *Pravi potomci* vozlišča u so vsi njegovi potomci razen samega vozlišča u.
- *Predniki* vozlišča u so vozlišče u sámo ter njegov starš, stari starš itd.
- *Pravi predniki* vozlišča u so vsi njegovi predniki razen samega vozlišča u.

Razpoznava prereznih točk

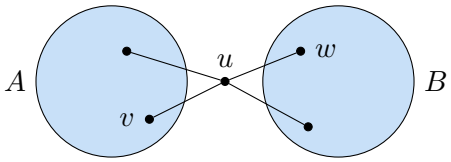
Zapisali bomo dve trditvi, ki določata potrebna in zadostna pogoja za to, da je določeno vozlišče prerezna točka.

Trditev 2. *Koren DFS-drevesa u je prerezna točka natanko tedaj, ko ima u v DFS-drevesu vsaj dva otroka.*

Dokaz. ($\Rightarrow$) Naj bo vozlišče u prerezna točka. To pomeni, da mora imeti v grafu vsaj dva soseda (če ima le enega, potem graf ne bo razpadel na dva dela, če u odstranimo), in to taka, da edina pot med njima poteka prek vozlišča u. Naj bosta v in w soseda vozlišča u s to lastnostjo.

Brez izgube splošnosti lahko predpostavimo, da je $v < w$, kar pomeni, da bo DFS najprej obiskal v in šele kasneje w. V trenutku, ko DFS obišče (torej posivi) v, je vozlišče u takisto sivo (jasno, v vozlišču u smo *sprožili* naš dragi DFS, saj je u koren DFS-drevesa), vozlišče w pa je še belo. Če je $v - u - w$ edina pot od v do w in če je u sivo, potem ne obstaja bela pot od v do w, zato po izreku o beli poti w ne postane potomec v-ja v DFS-drevesu. Edina možnost je torej ta, da w postane neposredni otrok u-ja (do w-ja bomo prišli šele takrat, ko se bo rekurzija »odvila« in se bomo vrnili do u-ja). Vozlišče u ima potemtakem v DFS-drevesu vsaj dva otroka.

($\Leftarrow$) Enakovredna trditev se glasi takole: če koren u ni prerezna točka, potem ima v drevesu kvečjemu enega otroka. Privzemimo torej, da u ni prerezna točka in da so njegovi sosedje v grafu vozlišča $v_1 < v_2 < \ldots < v_k$. Ker u ni prerezna točka, za vsak par vozlišč v_i in v_j obstaja poleg poti $v_i - u - v_j$ še alternativna pot, ki ne poteka prek u. (Jasno: če u



Slika 3: Ilustracija trditve 3.

odstranimo, mora graf ostati povezan, zato mora biti mogoče od kateregakoli v_i priti do kateregakoli v_j, ne da bi prečkali u.) Ko DFS torej najde (in posivi) v_1, so vsa ostala vozlišča z izjemo u bela. To pomeni, da v trenutku odkritja vozlišča v_1 obstaja bela pot od v_1 do vseh ostalih sosedov u-ja, zato $v_2, \ldots, v_k$ postanejo potomci v_1 v DFS-drevesu. Od tod sledi, da bodo $v_1, \ldots, v_k$ sicer tudi *potomci* u-ja, vendar pa bo edini u-jev *otrok* vozlišče v_1. □

Trditev 3. *Naj bo u vozlišče, ki ni koren DFS-drevesa. Vozlišče u je prerezna točka natanko tedaj, ko ima v DFS-drevesu otroka s, tako da niti s niti noben pravi potomec vozlišča s nima povratne povezave do nekega* pravega *prednika vozlišča u.*

Dokaz. ($\Rightarrow$) Naj bo vozlišče u prerezna točka. To pomeni, da ima graf dva neprazna dela, recimo A in B (slika 3), ki sta povezana samo prek vozlišča u. Osredotočimo se na trenutek, ko DFS obišče vozlišče u. Ker u ni koren DFS-drevesa, je DFS do njega prispel prek enega od njegovih sosedov, recimo prek vozlišča v v delu A. V tistem trenutku je celoten del B še bel, saj je vozlišče u edini prehod med deloma A in B. Recimo, da je prvo vozlišče v delu B, ki ga DFS obišče, vozlišče w. Vozlišče w tako postane otrok vozlišča u v DFS-drevesu, vsa vozlišča v delu B pa (po izreku o belih poteh) potomci vozlišča w, vendar pa nobeden od njih ne bo imel povratne povezave do kateregakoli pravega prednika vozlišča u (vsi ti se namreč nahajajo v delu A). Iskani otrok s je torej vozlišče w.

($\Leftarrow$) Če u ni prerezna točka, potem med vsakim parom $v, w \in Adj[u]$ obstaja pot, ki se izogne vozlišču u. Ker u ni koren DFS-drevesa, bo DFS do njega prispel prek, recimo, soseda v. V trenutku odkritja vozlišča u bo vozlišče v torej sivo, w pa naj bo sosed vozlišča u, ki je takrat še bel (če takega soseda ni, potem u nima otrok v DFS-drevesu in lastnost iz trditve velja). Vozlišče v torej postane starš, vozlišče w pa otrok vozlišča u v DFS-drevesu. Ker obstaja pot od w do v, ki ne poteka prek u, jo bo DFS našel in pri tem ustvaril povratno povezavo med *nekim* potomcem vozliščem w (to je lahko tudi vozlišče w sámo) in *nekim* prednikom vozlišča v (to je lahko tudi vozlišče v sámo). □

Oglejmo si, ali se položaji prereznih točk na grafu s slike 1 skladajo s teorijo:

- Vozlišče A je koren DFS-drevesa in ima v DFS-drevesu enega samega otroka, zato ni prerezna točka.
- Vozlišče B ima v DFS-drevesu otroka C, ki nima povratne povezave do B-jevih pravih prednikov (to je tako ali tako samo A), prav tako pa nobeden od C-jevih pravih potomcev (teh tako ali tako ni) nima takšne povratne povezave. To pomeni, da je B prerezna točka.
- Vozlišče C v DFS-drevesu nima otrok, zato ni prerezna točka.
- Vozlišče D prav tako nima otrok.
- Vozlišče E ima otroke D, F in G. Potomstvo otroka D (torej samo D) ima povratno povezavo do E-jevega pravega prednika (do A). Enako velja za potomstvo otroka F (F in I): F ima povezavo do B. No, potomstvo vozlišča G (G in H) ima povratno

povezavo samo do samega E-ja, ne pa tudi do njegovih pravih prednikov. Vozlišče G je torej E-jev otrok, čigar potomstvo nima povratne povezave do pravih E-jevih prednikov, zato je E prerezna točka.

- Vozlišče F ima otroka I, čigar potomstvo (torej I) nima povratne povezave do F-jevih pravih prednikov. To pomeni, da je F prav tako prerezna točka.
- Vozlišče G ima enega samega otroka, njegovo potomstvo (H) pa ima povratno povezavo do G-jevega pravega prednika (E). To pomeni, da G ni prerezna točka.
- Vozlišče H nima otrok, zato ni prerezna točka.
- Vozlišče I prav tako nima otrok.

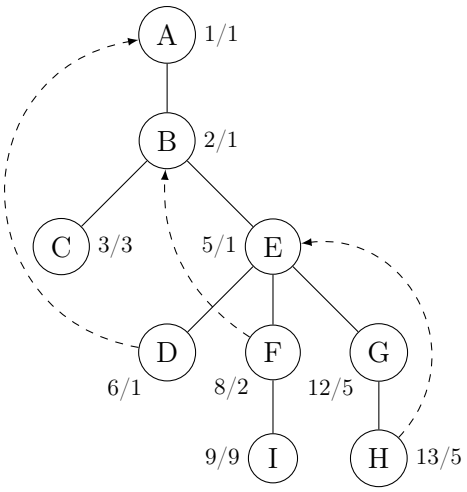
Dopolnitev algoritma DFS

Definirajmo

$v.low = \min(v.d, \min\{w.d \mid (u, w) \text{ je povratna povezava za nekega potomca } u \text{ vozlišča } v\})$.

Kaj pomeni vrednost $v.low$ v DFS-drevesu? Ker DFS-drevo gradimo hkrati z odkrivanjem vozlišč v grafu, imajo otroci poznejše čase odkritja ($v.d$) od staršev. Vrednost $v.low$ nam torej predstavlja najvišje ležeče vozlišče, do katerega pridemo prek ene same povratne povezave od potomstva vozlišča v (kar vključuje tudi vozlišče v sámo). Slika 4 prikazuje DFS-drevo s pripisanimi vrednostmi $v.d$ in $v.low$.

Kako lahko $v.low$ izkoristimo za naš problem? Oglejmo si, recimo, vozlišče E z $v.d = 5$. Njegovi otroci so D (D.$low = 1$), F (F.$low = 2$) in G (G.$low = 5$). Aha! Vozlišče E ima otroka G, čigar potomstvo ima povratno povezavo le do vozlišča E samega (G.$low = 5$), ne pa tudi do pravih prednikov vozlišča E. Zaradi obstoja otroka G je E torej prerezna točka; če bi E imel le otroka D (D.$low = 1$) in F (F.$low = 2$), ne bi bil prerezna točka.



Slika 4: Vrednosti $v.d$ in $v.low$ (v obliki $v.d/v.low$).

Sedaj nam je jasno:

- Če je vozlišče u koren DFS-drevesa, potem ni druge, kot da preštejemo njegove otroke v DFS-drevesu. Če ima vsaj dva otroka, je prerezna točka, sicer pa pač ni.
- Če vozlišče u ni koren DFS-drevesa, potem je prerezna točka natanko tedaj, ko ima otroka v, za katerega je $v.low \geqslant u.d$.

Super! Zdaj nam preostane le še to, da opisano pravilo vtkemo v naš dragi DFS. Ni tako težko:

- Ko vozlišče u odkrijemo, nastavimo $u.low \leftarrow u.d$ ($u.d$ je zgornja meja za $u.low$).
- Ko smo v vozlišču u in naletimo na sivega soseda v, najdemo povratno povezavo (u, v). Tedaj nastavimo $u.low \leftarrow \min(u.low, v.d)$. Na ta način bo vozlišče u že dobilo pravo vrednost $u.low$. Poskrbeti moramo le še za to, da bodo pravo vrednost $u.low$ dobili tudi njegovi predniki v DFS-drevesu.
- Ko smo v vozlišču u in najdemo belega soseda v, ustvarimo drevesno povezavo (u, v), obenem pa rekurzivno pokličemo DFS na vozlišču v. No, ko se vrnemo iz rekurzivnega klica, nastavimo $u.low \leftarrow \min(u.low, v.low)$ in na ta način zagotovimo, da se bo čas odkritja najvišjega vozlišča, dostopnega prek povratne povezave iz nekega vozlišča w, postopno (prek rekurzivnega sestopanja) prenesel na vse prednike vozlišča w.

To je to! Algoritem le še zapišimo (algoritem 2) in se poslovimo. No, skoraj. Vrednosti $v.f$ v resnici ne potrebujemo, pa tudi barv bi se lahko znebili, saj bi lahko na vprašanje, ali smo soseda v trenutnega vozlišča u že kdaj odkrili, odgovorili preprosto s preverjanjem vrednosti atributa $v.d$. Skratka, algoritem bi lahko bil še nekoliko krajši, a to ne moti velikih duhov.

Algoritem 2 Algoritem za iskanje prereznih točk.

```
function DFS-PREREZNE-TOČKE(u)
    u.barva ← siva
    čas ← čas + 1
    u.d ← čas
    u.low ← u.d
    for all v ∈ Adj[u] do
        if v.barva = bela then
            ▷ (u, v) je drevesna povezava ◁
            if u = 0 then                          ▷ DFS sprožimo iz vozlišča 0
                štOtrokKorena ← štOtrokKorena + 1
            v.predhodnik ← u
            DFS-PREREZNE-TOČKE(v)
            if v.low ⩾ u.d then
                PRINT(u)                           ▷ u je prerezna točka
            u.low ← min(u.low, v.low)
        else if v ≠ u.predhodnik then
            ▷ (u, v) je povratna povezava ◁
            u.low ← min(u.low, v.d)
    čas ← čas + 1
    u.f ← čas
    u.barva ← črna

▷ V glavnem delu programa, npr. funkciji main: ◁
for v ← 0 to n − 1 do
    v.barva ← bela
    v.predhodnik ← −1
    v.d ← −1
    v.f ← −1
    v.low ← ∞
čas ← 0
štOtrokKorena ← 0
DFS-PREREZNE-TOČKE(0)
if štOtrokKorena ⩾ 2 then
    PRINT(0)
```