Devete vaje APS2: Maksimalni pretoki

1 Problem

Podano je omrežje (usmerjeni graf) z vozlišči $0, 1, \ldots, n-1$, pri čemer vozlišče 0 imenujemo *izvor*, vozlišče $n-1$ pa *ponor*. Graf ima nenegativno utežene povezave, uteži pa imenujemo *kapacitete*. Naj c_{ij} označuje kapaciteto usmerjene povezave (i, j). Če vozlišči u in v nista povezani, si lahko predstavljamo, da je $c_{uv} = 0$.

Cilj je določiti takšne *pretoke* x_{ij}, da bo

- $0 \leq x_{ij} \leq c_{ij}$ za vsako povezavo (i, j) (pretok ne sme biti večji od kapacitete);
- $\sum_j x_{ji} = \sum_j x_{ij}$ za vsak $i \in \{1, \ldots, n-2\}$ (toliko snovi, kot je v vozlišče priteče, je iz vozlišča tudi odteče);
- $\sum_j x_{0j} = \sum_j x_{j,n-1}$ (toliko snovi, kot se je v izvoru poraja, je v ponoru ponikne);
- vrednost $v = \sum_j x_{0j} = \sum_j x_{j,n-1}$ (*pretok po omrežju*) maksimalna možna.

2 Algoritem

Za začetek predpostavimo, da omrežje nima dvosmernih povezav, torej da za vsak par (i, j) obstaja kvečjemu ena od povezav (i, j) in (j, i).

Naj bo *pot* od izvora do ponora zaporedje $\langle 0, u_1, \ldots, u_k, n-1 \rangle$, pri čemer sta $(0, u_1)$ in $(u_k, n-1)$ povezavi v omrežju, poleg tega pa za vsak $i \in \{1, \ldots, k-1\}$ obstaja bodisi povezava (u_i, u_{i+1}) bodisi povezava (u_{i+1}, u_i). V nasprotju z običajno definicijo poti je torej lahko pot v kontekstu iskanja maksimalnih pretokov sestavljena tako iz povezav v pravilni smeri kot iz povezav v obratni smeri.

Pot je *nezasičena*, če so vse povezave na njej nezasičene. Povezava (i, j) je *nezasičena v pravilni smeri*, če je $x_{ij} < c_{ij}$. Povezava (i, j) je *nezasičena v obratni smeri* (od j do i), če je $x_{ij} > 0$. V prvem primeru lahko pretok po povezavi povečamo za $\delta_{ij} = c_{ij} - x_{ij}$, v drugem pa ga lahko *zmanjšamo* za $\delta_{ij} = x_{ij}$ (pretok v obratni smeri nam dejansko škoduje).

Nezasičeno pot $\langle 0, u_1, \ldots, u_k, n-1 \rangle$ *zasitimo* tako, da izračunamo $\delta = \min\{\delta_{0,u_1}, \delta_{u_1,u_2}, \ldots, \delta_{u_{k-1},u_k}, \delta_{u_k,n-1}\}$, nato pa vsaki povezavi v pravilni smeri pretok povečamo za δ, vsaki povezavi v obratni smeri pa pretok zmanjšamo za δ. Na ta način bomo ohranili vse zahteve iz prejšnjega razdelka, pretok po omrežju pa bomo povečali.

Algoritem za iskanje maksimalnega pretoka deluje enostavno tako, da vse pretoke nastavi na 0, nato pa v vsaki iteraciji poišče nezasičeno pot in jo zasiti. Algoritem se ustavi, ko ne najde več nobene nezasičene poti. Nezasičene poti lahko iščemo na različne načine, priporočljivo pa je uporabiti algoritem iskanja v širino (BFS), saj nam zagotavlja polinomsko časovno zahtevnost (ta znaša $O(VE^2)$, saj ena iteracija BFS traja $O(E)$ časa, število iteracij pa je v najslabšem primeru enako $O(VE)$).

Algoritem za iskanje nezasičene poti na osnovi BFS je prikazan kot algoritem 1. Algoritem označuje vozlišča z oznakami (p, t), kjer je p predhodnik trenutnega vozlišča na

nezasičeni poti, ki jo sestavljamo v trenutni iteraciji, t pa maksimalni pretok, ki ga lahko pripeljemo do trenutnega vozlišča. Ko označimo ponor, nam vrednost t pove, za koliko lahko povečamo pretok po pravkar odkriti nezasičeni poti, s sledenjem oznakam p pa lahko to pot rekonstruiramo (v smeri od ponora do izvora).

Algoritem 1 Algoritem za iskanje nezasičene poti na osnovi BFS

```
function poiščiNezasičenoPot(G = (V, E))
    vozlišče 0 označi z (/, ∞)
    dodaj vozlišče 0 na konec vrste
    while vrsta ni prazna do
        u := prvo vozlišče v vrsti
        t := druga komponenta oznake vozlišča u
        odstrani vozlišče u iz vrste
        for all s ∈ V such that vozlišče s še ni označeno ∧ ((u, s) ∈ E ∨ (s, u) ∈ E) do
            δ := 0
            if (u, s) ∈ E ∧ x_us < c_us then
                δ := min(t, c_us − x_us)
                vozlišče s označi z (u⁺, δ)
                dodaj vozlišče s na konec vrste
            else if (s, u) ∈ E ∧ x_su > 0 then
                δ := min(t, x_su)
                vozlišče s označi z (u⁻, δ)
                dodaj vozlišče s na konec vrste
            end if
            if δ > 0 ∧ s = n − 1 then
                return δ // našli smo nezasičeno pot; pretok po njej lahko povečamo za δ
            end if
        end for
    end while
    return 0 // ni več nezasičenih poti
end function
```

3 Primer

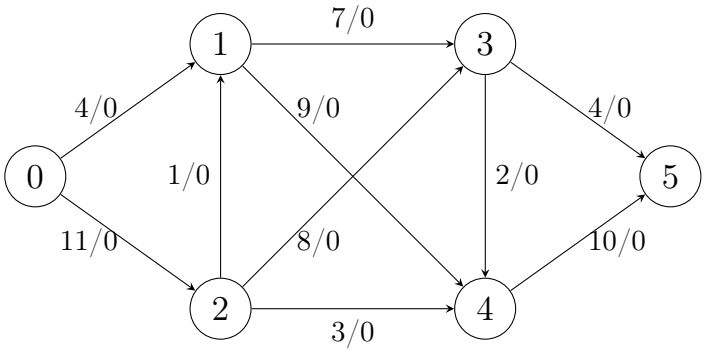
Na slikah 1–7 je prikazan primer izvajanja algoritma. Podrobneje si oglejmo izvajanje četrte iteracije (slika 5):

- Vozlišče 0 označimo z $(/, \infty)$ in ga dodamo v vrsto. Vsebina vrste je trenutno [0].
- Vozlišče 0 z oznako $(/, \infty)$ odvzamemo iz vrste. Njegova soseda sta 1 in 2, vendar pa povezavi (0, 1) pretoka ne moremo povečati, zato obravnavamo le vozlišče 2. Vozlišče 2 označimo z $(0^+, 7)$ in ga dodamo v vrsto. Vsebina vrste je trenutno [2].
- Vozlišče 2 z oznako $(0^+, 7)$ odvzamemo iz vrste. Njegovi sosedje (tako v pravilni kot v obratni smeri povezav) so 0, 1, 3 in 4. Sosed 0 je že označen, povezavi (2, 1) in (2, 4) pa sta že zasičeni, zato obravnavamo le vozlišče 3. Vozlišče 3 označimo z $(2^+, 7)$ $(7 = \min(7, 8 - 0))$ in ga dodamo v vrsto. Vsebina vrste je trenutno [3].
- Vozlišče 3 z oznako $(2^+, 7)$ odvzamemo iz vrste. Njegovi sosedje so 1, 2, 4 in 5, vendar pa obravnavamo le soseda 1 in 4. Vozlišče 1 označimo s $(3^-, 4)$ $(4 = \min(7, 4))$,

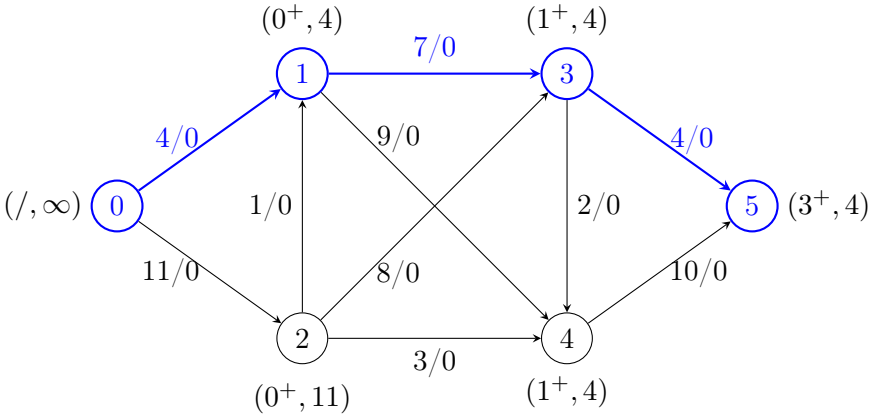
vozlišče 4 pa s $(3^+, 2)$. Obe vozlišči dodamo v vrsto. Vsebina vrste je trenutno [1, 4].

- Iz vrste odstranimo vozlišče 1, ki ima štiri sosede, vendar pa so vsi že označeni. Vsebina vrste je trenutno [4].
- Iz vrste odstranimo vozlišče 4 z oznako $(3^+, 2)$. Njegovega edinega neoznačenega soseda (vozlišče 5) označimo s $(4^+, 2)$ in ga dodamo v vrsto. Ker smo označili ponor, smo našli nezasičeno pot.
- Na podlagi prvih komponent oznak rekonstruiramo pot kot $5 \leftarrow 4 \leftarrow 3 \leftarrow 2 \leftarrow 0$. Pretok po poti povečamo za 2 (druga komponenta oznake ponora).

V zadnji iteraciji ugotovimo, da pretoka ne moremo več povečati, spotoma pa odkrijemo še *minimalni prerez* grafa — razbitje množice vozlišč na podmnožici S in T, tako da je $S \cap T = \emptyset$, $S \cup T = V$, $0 \in S$ in $n - 1 \in T$ ter da je vsota $\sum_{i \in S, j \in T} c_{ij}$ minimalna. Ta vsota je enaka maksimalnemu pretoku.



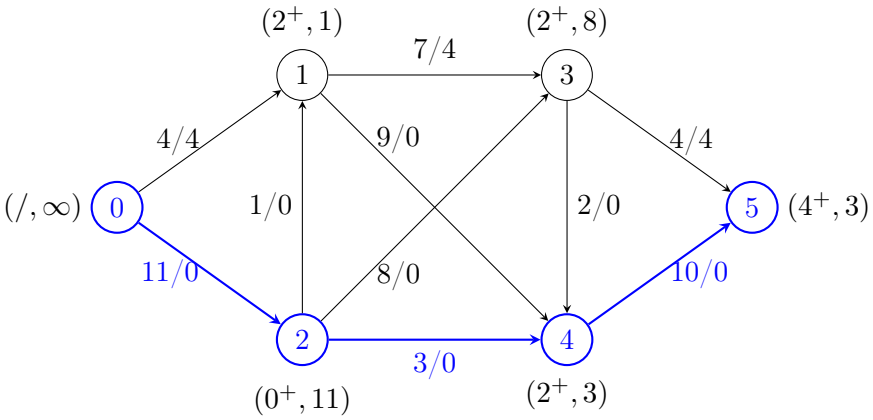
Slika 1: Začetno omrežje.



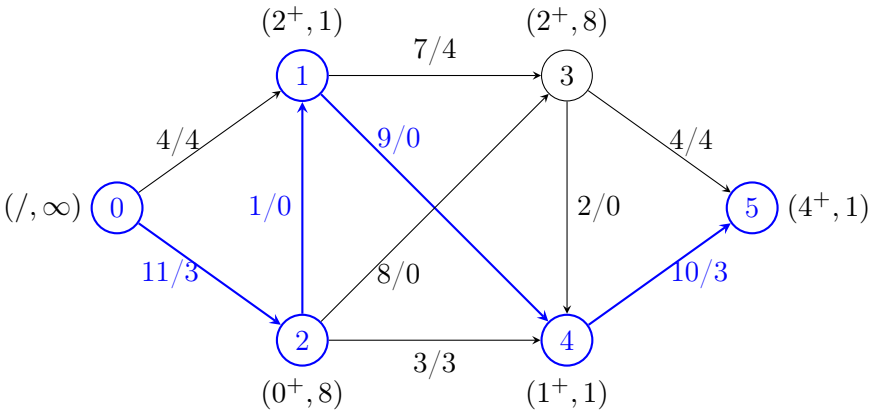
Slika 2: Prva iteracija: pretok po označeni poti se poveča za 4.

4 Obravnava dvosmernih povezav

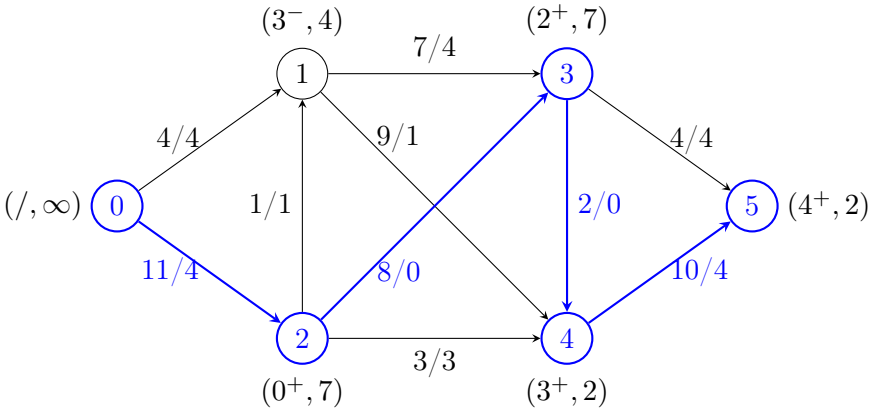
Dvosmerne povezave obravnavamo tako kot enosmerne, upoštevamo le, da pretok vedno teče le v eni smeri. Recimo, da je kapaciteta povezave (i, j) enaka a, kapaciteta povezave



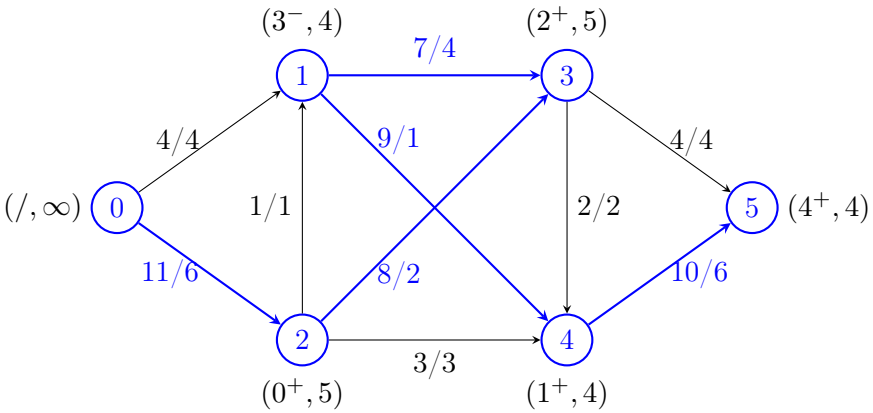
Slika 3: Druga iteracija: pretok po označeni poti se poveča za 3.



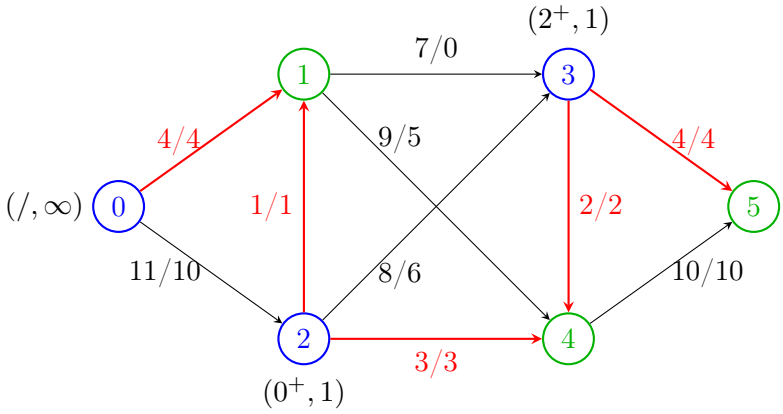
Slika 4: Tretja iteracija: pretok po označeni poti se poveča za 1.



Slika 5: Četrta iteracija: pretok po označeni poti se poveča za 2.



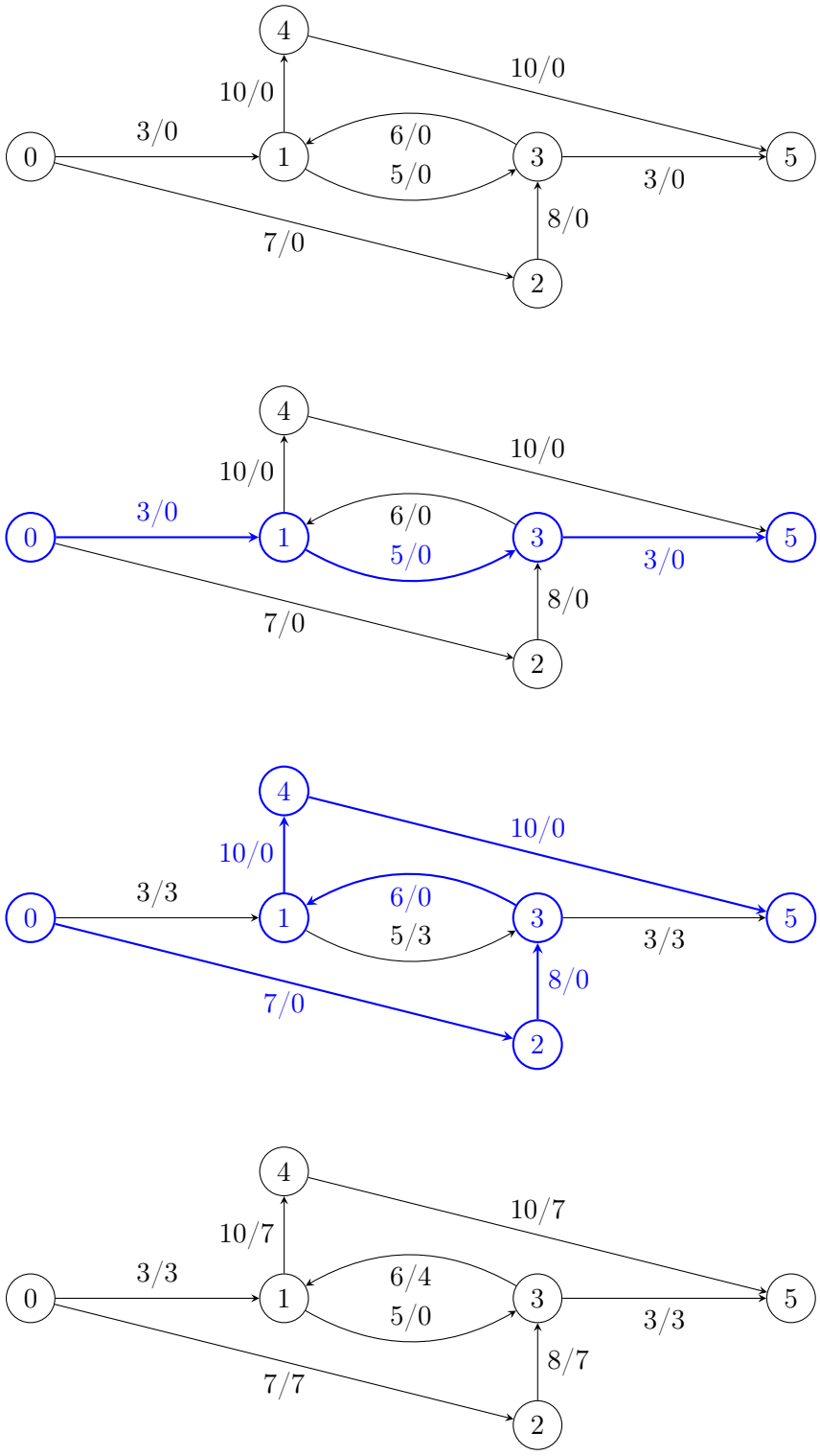
Slika 6: Peta iteracija: pretok po označeni poti se poveča za 4. Povezavo (1, 3) prehodimo v obratni smeri, zato se bo pretok po njej *zmanjšal* za 4.



Slika 7: Šesta iteracija: pretoka ne moremo več povečati. Označene povezave tvorijo minimalni prerez (modra vozlišča pripadajo množici S, zelena pa množici T).

(j, i) pa b in da trenutno teče $x \leq a$ enot v smeri $i \to j$. Pretok v smeri $j \to i$ lahko sedaj povečamo za $x + b$ enot (x enot za nevtralizacijo pretoka v smeri $i \to j$ in še b enot, ki jih dopušča povezava $j \to i$).

Oglejmo si primer na sliki 8. V prvi iteraciji zasitimo pot $0 \to 1 \to 3 \to 5$, v drugi pa odkrijemo nezasičeno pot $0 \to 2 \to 3 \to 1 \to 4 \to 5$. Povezavi $3 \to 1$, po kateri trenutno tečejo 3 enote v nasprotni smeri, bi lahko pretok povečali celo za 9 enot (3 enote za nevtralizacijo pretoka v nasprotni smeri in še 6 enot za zapolnitev kapacitete), vendar pa smo omejeni s 7 enotami, ki jih s seboj prinesemo iz vozlišča 0.



Slika 8: Drugi primer.