Desete vaje APS2: Topološko urejanje in najkrajše poti

1 Topološko urejanje

Topološko urejanje lahko uporabljamo na usmerjenih acikličnih grafih (DAG). Če od vozlišča u vodi pot do vozlišča v, potem je vozlišče u v topološkem vrstnem redu pred vozliščem v. Če ne obstaja niti pot od u do v niti pot od v do u, je lahko v topološkem vrstnem redu u pred v ali pa v pred u.

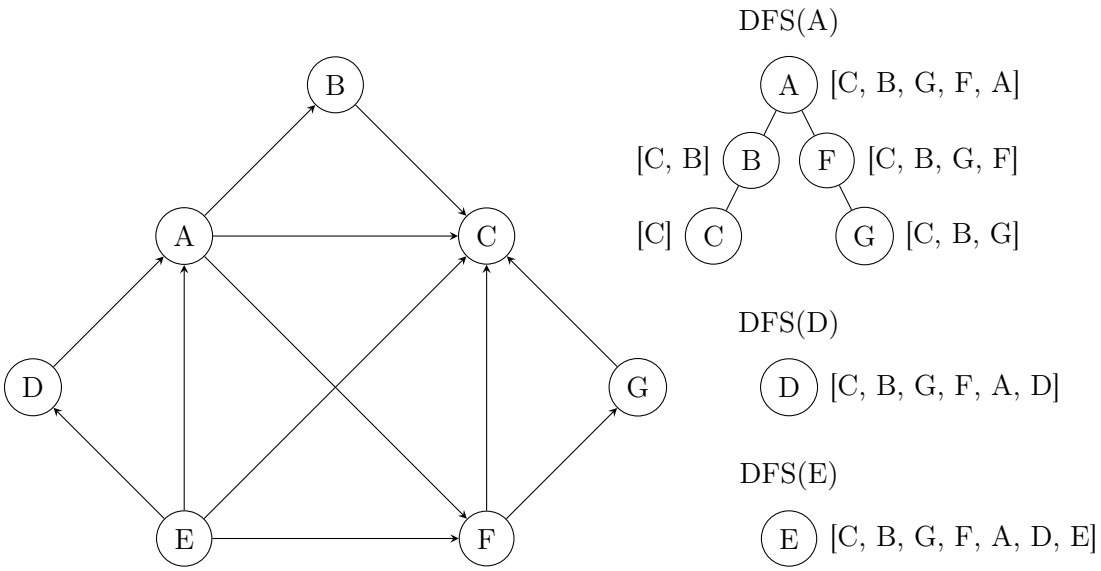
Topološko urejanje lahko izvedemo v času $O(E)$: poiščemo vozlišče z vhodno stopnjo 0 (pri DAG ga vedno lahko najdemo) in ga odstranimo (skupaj s pripadajočimi povezavami, seveda), nato pa ta postopek ponavljamo, dokler ne odstranimo vseh vozlišč. No, obstaja še Algoritem 1, ki temelji na iskanju v globino (DFS).

Algoritem 1 Algoritem za določanje topološkega vrstnega reda

```
function TOPOSORT
    t ← prazna tabela
    vsa vozlišča označi kot neobiskana
    for u ← 1, 2, ..., n do
        if vozlišče u ni obiskano then
            DFS(u, t)
        end if
    end for
    izpiši tabelo t v obratnem vrstnem redu
end function

function DFS(u, t)
    označi vozlišče u kot obiskano
    for all v ∈ sosedje(u) do
        if vozlišče v ni obiskano then
            DFS(v, t)
        end if
    end for
    dodaj vozlišče v v tabelo t
end function
```

Slika 1 prikazuje primer izvajanja algoritma. Najprej poženemo DFS na vozlišču A. V tabelo t bomo (v tem vrstnem redu) dodali vozlišča C, B, G, F in A. Nato iskanje v globino poženemo še na vozlišču D (v tabelo bomo dodali še D) in E (v tabelo bomo dodali še E). Topološki vrstni red je potemtakem E, D, A, F, G, B, C.



Slika 1: Določanje topološkega vrstnega reda.

2 Najkrajše poti

Na teh vajah se bomo ukvarjali le s problemom iskanja najkrajših poti od enega vozlišča (rekli mu bomo *izhodišče* in predpostavili, da ima številko 1) do vseh ostalih.

2.1 Brez negativno uteženih povezav: Dijkstra

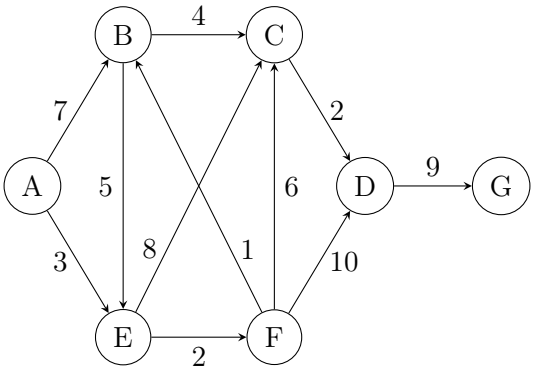
Če so vse razdalje (uteži na povezavah) nenegativne, lahko uporabimo *Dijkstrov algoritem*, ki teče v času $O(E \log V)$. Algoritem za vsako vozlišče v vzdržuje vrednost d_v, ki predstavlja trenutno znano zgornjo mejo najkrajše razdalje od izhodišča (vozlišča 1) do vozlišča v. Na začetku nastavimo d_1 na 0 in $d_v = \infty$ za vse $v \neq 1$, nato pa v vsaki iteraciji izmed vozlišč, ki jih še nismo izbrali, izberemo vozlišče u z najmanjšim d_u (izkaže se, da je d_u v tem trenutku *dejanska* najkrajša pot od 1 do u, ne le njena zgornja meja) in posodobimo razdalje do sosedov vozlišča u.

Slika 2 prikazuje primer poteka Dijkstrovega algoritma (izhodišče je vozlišče A). V preglednici je z zvezdico označeno trenutno izbrano vozlišče (na podlagi tega vozlišča se izračuna naslednja vrstica preglednice), vozlišča, ki smo jih izbrali v predhodnih iteracijah (in jih potemtakem ne upoštevamo več), pa so podčrtana.

Dijkstrov algoritem predpostavlja, da imajo vse povezave nenegativne uteži. Za graf na sliki 3 (vozlišče A je izhodišče) bo proizvedel napačen rezultat, saj bo vozlišče B izbral, ko bo (po njegovem »mnenju«) najkrajša razdalja do njega enaka 3, kasneje pa se na vozlišče B ne bo več vračal in tako ne bo ugotovil, da je razdalja do njega dejansko 2.

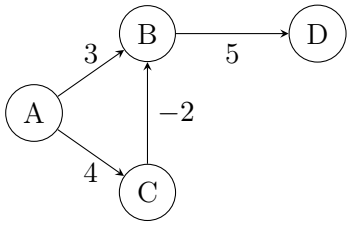
2.2 Z negativno uteženimi povezavami, a brez negativnih ciklov: Bellman-Ford

Če imamo v grafu negativno utežene usmerjene povezave, ne pa tudi ciklov z negativno vsoto razdalj ali dvosmernih negativno uteženih povezav, si lahko pomagamo z Bellman-Fordovim algoritmom. Ta zagotavlja pravilen rezultat tudi v takih primerih, a za ceno bistveno večje časovne zahtevnosti ($O(VE)$).



A	B	C	D	E	F	G
0^*	∞	∞	∞	∞	∞	∞
$\underline{0}$	7	∞	∞	3^*	∞	∞
$\underline{0}$	7	11	∞	$\underline{3}$	5^*	∞
$\underline{0}$	6^*	11	15	$\underline{3}$	$\underline{5}$	∞
$\underline{0}$	$\underline{6}$	10^*	15	$\underline{3}$	$\underline{5}$	∞
$\underline{0}$	$\underline{6}$	$\underline{10}$	12^*	$\underline{3}$	$\underline{5}$	∞
$\underline{0}$	$\underline{6}$	$\underline{10}$	$\underline{12}$	$\underline{3}$	$\underline{5}$	21

Slika 2: Primer izvedbe Dijkstrovega algoritma.



Slika 3: Graf, na katerem Dijkstra odpove, Bellman-Ford pa deluje.

Bellman-Fordov algoritem v h-ti iteraciji (za $h = 0, 1, \ldots, n-1$) za vsako vozlišče $i \in \{1, \ldots, n\}$ izračuna vrednost d_i^h, ki predstavlja dolžino najkrajše poti od vozlišča 1 do vozlišča i, ki je sestavljena iz kvečjemu h povezav. Vrednosti d_i^h računamo po formuli

$$d_i^h = \begin{cases} 0 & \text{v primeru } i = 1; \\ \infty & \text{v primeru } h = 0 \text{ in } i \neq 1; \\ \min(d_i^{h-1}, \min_{k \neq i}\{d_k^{h-1} + c_{ki}\}) & \text{sicer,} \end{cases}$$

pri čemer je c_{ij} razdalja po povezavi od i do j oziroma ∞, če povezava $i \to j$ ne obstaja. Pri iskanju najkrajše poti od vozlišča 1 do vozlišča i, ki vodi prek največ h povezav, upoštevamo že odkrito najkrajšo pot prek največ $h-1$ povezav, poleg tega pa še vse poti z največ $h-1$ povezavami od vozlišča 1 do tistih vozlišč k, od katerih vodi neposredna povezava do vozlišča i.

Koliko iteracij moramo izvesti? Ker je dolžina najdaljše aciklične poti v grafu z n vozlišči enaka $n-1$, je odgovor enak $n-1$, v praksi pa lahko algoritem prekinemo, če se v neki iteraciji nobena dolžina ne spremeni. Bellman-Fordov algoritem lahko uporabimo tudi za detekcijo negativnih ciklov: izvedemo še n-to iteracijo in če se katerakoli razdalja spremeni, ima graf negativen cikel.

Poženimo Bellman-Fordov algoritem na grafu s slike 3. Na začetku ($h = 0$) imamo $d_{\mathrm{A}}^0 = 0$ in $d_{\mathrm{B}}^0 = d_{\mathrm{C}}^0 = d_{\mathrm{D}}^0 = \infty$. Po prvi iteraciji velja

$$\begin{aligned} d_{\mathrm{A}}^1 &= 0; \\ d_{\mathrm{B}}^1 &= \min(d_{\mathrm{B}}^0, \min\{d_{\mathrm{A}}^0 + c_{\mathrm{AB}}, d_{\mathrm{C}}^0 + c_{\mathrm{CB}}\}) = \min(\infty, \min\{0+3, \infty-2\}) = 3; \\ d_{\mathrm{C}}^1 &= \min(d_{\mathrm{C}}^0, \min\{d_{\mathrm{A}}^0 + c_{\mathrm{AC}}\}) = \min(\infty, \min\{0+4\}) = 4; \\ d_{\mathrm{D}}^1 &= \min(d_{\mathrm{D}}^0, \min\{d_{\mathrm{B}}^0 + c_{\mathrm{BD}}\}) = \min(\infty, \min\{\infty+5\}) = \infty. \end{aligned}$$

To so dolžine najkrajših poti s kvečjemu eno povezavo od vozlišča A do vseh ostalih vozlišč.

Po drugi iteraciji velja

$$
\begin{aligned}
d^2_{\mathrm{A}} &= 0;\\
d^2_{\mathrm{B}} &= \min(d^1_{\mathrm{B}}, \min\{d^1_{\mathrm{A}} + c_{\mathrm{AB}}, d^1_{\mathrm{C}} + c_{\mathrm{CB}}\}) = \min(3, \min\{0+3, 4-2\}) = 2;\\
d^2_{\mathrm{C}} &= \min(d^1_{\mathrm{C}}, \min\{d^1_{\mathrm{A}} + c_{\mathrm{AC}}\}) = \min(4, \min\{0+4\}) = 4;\\
d^2_{\mathrm{D}} &= \min(d^1_{\mathrm{D}}, \min\{d^1_{\mathrm{B}} + c_{\mathrm{BD}}\}) = \min(\infty, \min\{3+5\}) = 8.
\end{aligned}
$$

Po tretji (zadnji) iteraciji imamo

$$
\begin{aligned}
d^3_{\mathrm{A}} &= 0;\\
d^3_{\mathrm{B}} &= \min(d^2_{\mathrm{B}}, \min\{d^2_{\mathrm{A}} + c_{\mathrm{AB}}, d^2_{\mathrm{C}} + c_{\mathrm{CB}}\}) = \min(3, \min\{0+3, 4-2\}) = 2;\\
d^3_{\mathrm{C}} &= \min(d^2_{\mathrm{C}}, \min\{d^2_{\mathrm{A}} + c_{\mathrm{AC}}\}) = \min(4, \min\{0+4\}) = 4;\\
d^3_{\mathrm{D}} &= \min(d^2_{\mathrm{D}}, \min\{d^2_{\mathrm{B}} + c_{\mathrm{BD}}\}) = \min(8, \min\{2+5\}) = 7.
\end{aligned}
$$

Poženimo Bellman-Fordov algoritem še na grafu s slike 2. Tabela 1 prikazuje dolžine najkrajših poti od izhodišča A do ostalih vozlišč po posameznih iteracijah, tabela 2 pa dejanske najkrajše poti, sestavljene iz kvečjemu h povezav. Te lahko rekonstruiramo tako, da med delovanjem algoritma za vsako vozlišče hranimo njegovega predhodnika na najkrajši poti.

Tabela 1: Delovanje Bellman-Fordovega algoritma na grafu s slike 2. Tabela prikazuje dolžine najkrajših poti od vozlišča A do vseh ostalih po posameznih iteracijah.

h	A	B	C	D	E	F	G
0	0	∞	∞	∞	∞	∞	∞
1	0	7	∞	∞	3	∞	∞
2	0	7	11	∞	3	5	∞
3	0	6	11	13	3	5	∞
4	0	6	10	13	3	5	22
5	0	6	10	12	3	5	22
6	0	6	10	12	3	5	21

Tabela 2: Delovanje Bellman-Fordovega algoritma na grafu s slike 2. Tabela prikazuje dejanske najkrajše poti od vozlišča A do vseh ostalih po posameznih iteracijah.

h	A	B	C	D	E	F	G
0	A	–	–	–	–	–	–
1	A	AB	–	–	AE	–	–
2	A	AB	ABC	–	AE	AEF	–
3	A	AEFB	ABC	ABCD	AE	AEF	–
4	A	AEFB	AEFBC	ABCD	AE	AEF	ABCDG
5	A	AEFB	AEFBC	AEFBCD	AE	AEF	ABCDG
6	A	AEFB	AEFBC	AEFBCD	AE	AEF	AEFBCDG