14. Transitivní uzávěry

V předchozí kapitole jsme se zabývali algoritmy pro hledání nejkratších cest z daného počátečního vrcholu. Nyní se zaměříme na případy, kdy nás zajímají vzdálenosti, případně pouhá dosažitelnost, mezi všemi dvojicemi vrcholů.

Výstupem takového algoritmu bude *matice vzdáleností* (případně *matice dosažitelnosti*). Na ni se také můžeme dívat jako na *transitivní uzávěr* zadaného grafu – to je graf na téže množině vrcholů, jehož hrany odpovídají nejkratším cestám v grafu původním.

Jeden způsob, jak transitivní uzávěr spočítat, se ihned nabízí: postupně spustit Dijkstrův algoritmus pro všechny možné volby počátečního vrcholu, případně se před tím ještě zbavit záporných hran pomocí potenciálů. Tak dosáhneme časové složitosti $\mathcal{O}(mn + n^2 \log n)$. To je pro řídké grafy nejlepší známý výsledek – jen $\mathcal{O}(\log n)$-krát pomalejší, než je velikost výstupu.

Je-li graf hustý, pracují obvykle rychleji algoritmy založené na maticích, zejména na jejich násobení. Několik algoritmů tohoto druhu si nyní předvedeme, a to jak pro výpočet vzdáleností, tak pro dosažitelnost.

Graf na vstupu bude vždy zadán maticí délek hran – to je matice $n \times n$, jejíž řádky i sloupce jsou indexované vrcholy a na pozici (i, j) se nachází délka hrany ij; případné chybějící hrany doplníme s délkou $+\infty$.

Floydův-Warshallův algoritmus

Začněme algoritmem, který nezávisle na sobě objevili Floyd a Warshall. Funguje pro libovolný orientovaný graf bez záporných cyklů.

Označme D^k_{ij} délku nejkratší cesty z vrcholu i do vrcholu j přes vrcholy 1 až k (tím myslíme, že všechny vnitřní vrcholy cesty leží v množině $\{1, \ldots, k\}$). Jako obvykle položíme $D^k_{ij} = +\infty$, pokud žádná taková cesta neexistuje. Pak platí:

$$\begin{aligned} D^0_{ij} &= \text{délka hrany } ij, \\ D^n_{ij} &= \text{hledaná vzdálenost z } i \text{ do } j, \\ D^k_{ij} &= \min(D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj}). \end{aligned}$$

První dvě rovnosti plynou přímo z definice. Třetí rovnost dostaneme rozdělením cest z i do j přes 1 až k na ty, které se vrcholu k vyhnou (a jsou tedy cestami přes 1 až $k-1$), a ty, které ho použijí – každou takovou můžeme složit z cesty z i do k a cesty z k do j, obojí přes 1 až $k-1$.

Zbývá vyřešit jednu maličkost: složením cesty z i do k s cestou z k do j nemusí nutně vzniknout cesta, protože se nějaký vrchol může opakovat. V grafech bez záporných cyklů nicméně takový sled nemůže být kratší než nejkratší cesta, takže tím falešné řešení nevyrobíme. (Přesněji: ze sledu $i\alpha v\beta k\gamma v\delta j$, kde $v \notin \beta, \gamma$, můžeme vypuštěním části $v\beta k\gamma v$ nezáporné délky získat sled z i do j přes 1 až $k-1$, jehož délka nemůže být menší než D^{k-1}_{ij}.)

Samotný algoritmus postupně počítá matice $D^0, D^1, \ldots, D^n$ podle uvedeného předpisu:

1. $D^0 \leftarrow$ matice délek hran.
2. Pro $k = 1, \ldots, n$:
3. Pro $i, j = 1, \ldots, n$:
4. $D^k_{ij} \leftarrow \min(D^{k-1}_{ij}, D^{k-1}_{ik} + D^{k-1}_{kj})$.
5. Matice vzdáleností $\leftarrow D^n$.

Časová složitost tohoto algoritmu činí $\Theta(n^3)$. Kubickou prostorovou složitost můžeme snadno snížit na kvadratickou: Buď si uvědomíme, že v každém okamžiku potřebujeme jen aktuální matici D^k a předchozí D^{k-1}. Anebo nahlédneme, že můžeme D^{k-1} na D^k přepisovat na místě. U hodnot D_{ik} a D_{kj} je totiž podle definice stará i nová hodnota stejná. Algoritmu tedy stačí jediné pole velikosti $n \times n$, které na počátku výpočtu obsahuje vstup a na konci výstup.

Regulární výrazy

Předchozí algoritmus lze zajímavě zobecnit, totiž tak, aby pro každou dvojici vrcholů sestrojil vhodnou reprezentaci množiny všech sledů vedoucích mezi nimi. Tato reprezentace bude velice podobná regulárním výrazům známým z UNIXu a z teorie automatů. Budeme připouštět orientované multigrafy se smyčkami a násobnými hranami.

Definice: *Svazek* je množina sledů, které mají společný počáteční a koncový vrchol. *Typem* svazku nazveme uspořádanou dvojici těchto vrcholů. Místo „svazek typu (u, v)“ budeme obvykle říkat prostě *uv-svazek.*

Triviálními případy svazků jsou prázdná množina $\emptyset$, samotná hrana uv a pro $u = v$ také sled ε_u nulové délky. Svazky můžeme kombinovat těmito operacemi:

- $A \cup B$ – *sjednocení* dvou svazků téhož typu,
- AB nebo $A \cdot B$ – *zřetězení* uv-svazku A s vw-svazkem B: výsledkem je uw-svazek obsahující všechna spojení sledu z A se sledem z B,
- A^* – *iterace* uu-svazku: výsledkem je uu-svazek $\varepsilon \cup A \cup AA \cup AAA \cup \ldots$ (tedy všechna možná spojení konečně mnoha sledů z A).

Ve výrazu definujícím A^* jsme využili, že operace sjednocení a zřetězení jsou asociativní, takže je nemusíme závorkovat. Navíc sjednocení má ve výrazech nižší prioritu než zřetězení.

Svazky budeme obvykle reprezentovat *sledovými výrazy,* což jsou výrazy konečné délky sestávající z triviálních svazků a výše uvedených operací.

Pozorování: Sledy můžeme reprezentovat řetězci nad abecedou, jejíž symboly jsou identifikátory hran. Sledové výrazy pak odpovídají regulárním výrazům nad touto abecedou.

Ukážeme, jak pro všechny dvojice vrcholů i, j sestrojit sledový výraz R_{ij} popisující svazek všech sledů z i do j. Podobně jako u Floydova-Warshallova algoritmu

zavedeme R_{ij}^k coby výraz popisující sledy z i do j přes 1 až k a nahlédneme, že platí:

$$R_{ij}^0 = \begin{cases} \text{množina všech hran z } i \text{ do } j & \text{pokud } i \neq j \\ \varepsilon_i \cup \text{všechny smyčky v } i & \text{pokud } i = j \end{cases}$$
$$R_{ij}^n = \text{hledané } R_{ij},$$
$$R_{ij}^k = R_{ij}^{k-1} \cup R_{ik}^{k-1}(R_{kk}^{k-1})^* R_{kj}^{k-1}.$$

První dvě rovnosti opět plynou přímo z definice (množinu všech hran zapíšeme buď jako prázdnou nebo ji vytvoříme sjednocováním z jednoprvkových množin). Třetí rovnost vychází z toho, že každý sled z i do j buď neobsahuje k, nebo ho můžeme rozdělit na části mezi jednotlivými návštěvami vrcholu k.

Algoritmus tedy bude postupně budovat matice $R^0, R^1, \ldots, R^n$ podle těchto pravidel. Provedeme při tom $\Theta(n^3)$ operací, ovšem s výrazy, jejichž délka může být až řádově 4^n. Raději než jako řetězce je proto budeme ukládat v podobě acyklických orientovaných grafů: vrcholy budou operace, hrany je budou připojovat k operandům.

K přímému použití se takový exponenciálně dlouhý výraz hodí málokdy, ale může nám pomoci odpovídat na různé otázky týkající se sledů s danými koncovými vrcholy. Máme-li nějakou funkci f ohodnocující množiny sledů kódované sledovými výrazy, stačí umět vyhodnotit:

- výsledek pro triviální výrazy $f(\emptyset)$, $f(\varepsilon)$ a $f(e)$ pro hranu e,
- hodnoty $f(\alpha \cup \beta)$, $f(\alpha\beta)$ a $f(\alpha^*)$, známe-li již $f(\alpha)$ a $f(\beta)$.

Pro výpočet všech $f(R_{ij})$ nám pak stačí $\Theta(n^3)$ vyhodnocení funkce f.

Poznámka pro ctitele algebry: Výše uvedená konstrukce není nic jiného, než popis homomorfismu f z algebry $(\mathcal{S}, \emptyset, \varepsilon_1, \ldots, \varepsilon_n, e_1, \ldots, e_m, \cup, \cdot, ^*)$ nad množinou $\mathcal{S}$ všech svazků do nějaké algebry $(X, \mathbf{0}, c_1, \ldots, c_n, h_1, \ldots, h_m, \oplus, \otimes, \circledast)$, kde X je množina všech ohodnocení sledů, $\mathbf{0}$, $c_1, \ldots, c_n$ a $h_1, \ldots, h_m$ jsou konstanty, $\oplus$ a $\otimes$ binární operace a $\circledast$ unární operace. Průběh výpočtu upraveného algoritmu je pak homomorfním obrazem průběhu výpočtu původního algoritmu pracujícího přímo se svazky.

Příklady:

Nejkratší sled:

$$f(\emptyset) = +\infty$$
$$f(\varepsilon) = 0$$
$$f(e) = \ell(e) \quad (\text{délka hrany } e)$$
$$f(\alpha \cup \beta) = \min(f(\alpha), f(\beta))$$
$$f(\alpha\beta) = f(\alpha) + f(\beta)$$
$$f(\alpha^*) = \begin{cases} 0 & \text{pro } f(\alpha) \geq 0 \\ -\infty & \text{pro } f(\alpha) < 0 \end{cases}$$

(Pokud předpokládáme, že v grafu nejsou záporné cykly, je $f(\alpha^*)$ vždy nulové a dostaneme přesně Floydův-Warshallův algoritmus.)

Nejširší cesta (hranám jsou přiřazeny šířky, šířka cesty je minimum z šířek hran):

$$\begin{aligned} f(\emptyset) &= 0 \\ f(\varepsilon) &= \infty \\ f(e) &= w(e) \quad \text{(šířka hrany } e\text{)} \\ f(\alpha \cup \beta) &= \max(f(\alpha), f(\beta)) \\ f(\alpha\beta) &= \min(f(\alpha), f(\beta)) \\ f(\alpha^*) &= \infty \end{aligned}$$

Převod konečného automatu na regulární výraz: Vrcholy multigrafu budou odpovídat stavům automatu, hrany možným přechodům mezi stavy. Každou hranu ohodnotíme symbolem abecedy, po jehož přečtení automat přechod provede. Funkce f pak přiřadí uv-svazku ψ regulární výraz popisující množinu všech řetězců, po jejichž přečtení automat přejde ze stavu u do stavu v po přechodech z množiny ψ. Vyhodnocování funkce f odpovídá přímočarým operacím s řetězci.

Násobení matic

Řešíme-li grafové problémy pomocí matic, nabízí se použít známé subkubické algoritmy pro lineárně algebraické úlohy. Ty jsou obvykle založeny na efektivním násobení matic – dvě matice $n \times n$ můžeme vynásobit v čase:

- $\mathcal{O}(n^3)$ podle definice,
- $\mathcal{O}(n^{2.808})$ Strassenovým algoritmem [6],
- $\mathcal{O}(n^{2.376})$ algoritmem Coppersmithe a Winograda [3],
- $\mathcal{O}(n^{2.373})$ algoritmem Williamsové [7].

Obecně přijímaná hypotéza říká, že pro každé $\omega > 2$ existuje algoritmus násobící matice se složitostí $\mathcal{O}(n^\omega)$. Jediný známý dolní odhad je přitom $\Omega(n^2 \log n)$ pro aritmetické obvody s omezenou velikostí konstant [4]. Blíže se o tomto fascinujícím tématu můžete dočíst v Szegedyho článku [2].

Předpokládejme tedy, že umíme násobit matice v čase $\mathcal{O}(n^\omega)$ pro nějaké $\omega < 3$.

Co se stane, když mocníme matici sousednosti A grafu? V matici A^k se na pozici i, j nachází počet sledů délky k, které vedou z vrcholu i do vrcholu j. (Snadný důkaz indukcí vynecháváme.) Pro libovolné $k \geq n - 1$ jsou tedy v $(A + E)^k$ (kde E značí jednotkovou matici; doplnili jsme tedy do grafu smyčky) nenuly přesně tam, kde z i do j vede cesta.

To nám dává jednoduchý algoritmus pro výpočet matice dosažitelnosti R (R_{ij} je 1 nebo 0 podle toho, zda z i do j vede cesta). Do matice A doplníme na diagonálu jedničky a poté ji $\lceil \log_2 n \rceil$-krát umocníme na druhou. Abychom se vyhnuli velkým číslům, nahradíme po každém umocnění nenuly jedničkami. Celkem tedy provedeme $\mathcal{O}(\log n)$ násobení matic, což trvá $\mathcal{O}(n^\omega \log n)$.

Na tento postup se můžeme dívat i obecněji:

Definice: $(\oplus, \otimes)$-*součin* matic $A, B \in X^{n \times n}$, kde $\oplus$ a $\otimes$ jsou dvě asociativní binární operace na množině X, je matice C taková, že

$$C_{ij} = \bigoplus_{k=1}^{n} A_{ik} \otimes B_{kj}.$$

Klasické násobení matic je tedy $(+, \cdot)$-součin.

Pozorování: Pokud A a B jsou matice svazků (A_{ij} a B_{ij} jsou ij-svazky) a C jejich $(\cup, \cdot)$-součin, pak C_{ij} je svazek všech sledů vzniklých spojením nějakého sledu z A začínajícího v i a nějakého sledu z B končícího v j.

Podobně jako v předchozí sekci si tedy můžeme pořídit funkci f přiřazující svazkům hodnoty z nějaké množiny X a operace $\oplus$ a $\otimes$ na X, pro něž platí $f(\alpha \cup \beta) = f(\alpha) \oplus f(\beta)$ a $f(\alpha\beta) = f(\alpha) \otimes f(\beta)$. Pak stačí vzít matici popisující ohodnocení všech sledů délky 0 nebo 1 (to je obdoba matice sousednosti), a provést $\mathcal{O}(\log n)$ $(\oplus, \otimes)$-součinů k tomu, abychom znali ohodnocení svazků sledů délky právě k pro nějaké $k \geq n$.

S $(\vee, \wedge)$-součiny a maticí sousednosti s jedničkami na diagonále získáme algoritmus pro výpočet dosažitelnosti. Každý součin přitom můžeme provést jako obyčejné násobení matic následované přepsáním nenul na jedničky, takže celý výpočet běží v čase $\mathcal{O}(n^\omega \log n)$.

Podobně můžeme počítat i matici vzdáleností: začneme s maticí délek hran doplněnou o nuly na diagonále a použijeme $(\min, +)$-součiny. Tyto součiny ale bohužel neumíme převést na klasické násobení matic. Přesto je známo několik algoritmů efektivnějších než $\Theta(n^3)$, byť pouze o málo: například Zwickův [9] v čase $\mathcal{O}(n^3 \sqrt{\log \log n} / \log n)$ (založený na dekompozici a předpočítání malých bloků) nebo Chanův [1] v $\mathcal{O}(n^3 / \log n)$ (používající geometrické techniky). Abychom porazili Floydův-Warshallův algoritmus, potřebovali bychom ovšem větší než logaritmické zrychlení, protože součinů potřebujeme vypočítat logaritmicky mnoho.

Dodejme ještě, že pro grafy ohodnocené malými celými čísly je možné využít celou řadu dalších triků. Zájemce o tento druh algoritmů odkazujeme na Zwickův článek [8].

Rozděl a panuj

Předchozí převod je ovšem trochu marnotratný. Šikovným použitím metody Rozděl a panuj můžeme časovou složitost ještě snížit. Postup předvedeme pro dosažitelnost: na vstupu tedy dostaneme matici sousednosti A, výstupem má být její transitivní uzávěr A^* (matice dosažitelnosti). Všechny součiny matic v tomto oddílu budou typu $(\vee, \wedge)$.

Vrcholy grafu rozdělíme na dvě množiny X a Y přibližně stejné velikosti, bez újmy na obecnosti tak, aby matice A měla následující blokový tvar:

$$A = \begin{pmatrix} P & Q \\ R & S \end{pmatrix},$$

kde podmatice P popisuje hrany z X do X, podmatice Q hrany z X do Y, atd.

Věta: Pokud matici A^* zapíšeme rovněž v blokovém tvaru:

$$A^* = \begin{pmatrix} I & J \\ K & L \end{pmatrix},$$

bude platit:

$$\begin{aligned} I &= (P \vee QS^*R)^*, \\ J &= IQS^*, \\ K &= S^*RI, \\ L &= S^* \vee S^*RIQS^*. \end{aligned}$$

Důkaz: Jednotlivé rovnosti můžeme číst takto:

- **I:** Sled z X do X vznikne opakováním částí, z nichž každá je buďto hrana uvnitř X nebo přechod po hraně z X do Y následovaný sledem uvnitř Y a přechodem zpět do X.
- **J:** Sled z X do Y můžeme rozdělit v místě, kdy naposledy přechází po hraně z X do Y. První část přitom bude sled z X do X, druhá sled uvnitř Y.
- **K:** Se sledem z Y do X naložíme symetricky.
- **L:** Sled z Y do Y vede buďto celý uvnitř Y, nebo ho můžeme rozdělit na prvním přechodu z Y do X a posledním přechodu z X do Y. Část před prvním přechodem povede celá uvnitř Y, část mezi přechody bude tvořit sled z X do X a konečně část za posledním přechodem zůstane opět uvnitř Y. ♡

Algoritmus: Výpočet matice A^* provedeme podle předchozí věty. Spočítáme 2 tranzitivní uzávěry matic poloviční velikosti rekurzivním voláním, dále pak $\mathcal{O}(1)$ $(\vee, \wedge)$-součinů a $\mathcal{O}(1)$ součtů matic.

Časová složitost $t(n)$ tohoto algoritmu bude splňovat následující rekurenci:

$$t(1) = \Theta(1), \quad t(n) = 2t(n/2) + \Theta(1) \cdot \mu(n/2) + \Theta(n^2),$$

kde $\mu(k)$ značí čas potřebný na jeden $(\vee, \wedge)$-součin matic $k \times k$. Jelikož jistě platí $\mu(n/2) = \Omega(n^2)$, má tato rekurence podle kuchařkové věty řešení $t(n) = \mu(n)$.

Ukázali jsme tedy, že výpočet matice dosažitelnosti je nejvýše stejně náročný jako $(\vee, \wedge)$-násobení matic – můžeme ho proto provést v čase $\mathcal{O}(n^\omega)$. Dokonce existuje přímočarý převod v opačném směru, takže oba problémy jsou asymptoticky stejně těžké.

Podobně nalézt matici vzdáleností je stejně těžké jako $(\min, +)$-součin, takže na to stačí čas $\mathcal{O}(n^3/\log n)$.

Seidelův algoritmus

Pro neorientované neohodnocené grafy můžeme dosáhnout ještě lepších výsledků. Matici vzdáleností lze spočítat v čase $\mathcal{O}(n^\omega \log n)$ Seidelovým algoritmem [5]. Ten funguje následovně:

Definice: *Druhá mocnina grafu* G je graf G^2 na téže množině vrcholů, v němž jsou vrcholy i a j spojeny hranou právě tehdy, existuje-li v G sled délky nejvýše 2 vedoucí z i do j.

Pozorování: Matici sousednosti grafu G^2 získáme z matice sousednosti grafu G jedním $(\vee, \wedge)$-součinem, tedy v čase $\mathcal{O}(n^\omega)$.

Seidelův algoritmus bude postupovat rekurzivně: Sestrojí graf G^2, rekurzí spočítá jeho matici vzdáleností D' a z ní pak rekonstruuje matici vzdáleností D zadaného grafu. Rekurze končí, pokud $G^2 = G$ – tehdy je každá komponenta souvislosti zahuštěna na úplný graf, takže matice vzdáleností je rovna matici sousednosti.

Zbývá ukázat, jak z matice D' spočítat matici D. Zvolme pevně i a zaměřme se na funkce $d(v) = D_{iv}$ a $d'(v) = D'_{iv}$. Jistě platí $d'(v) = \lceil d(v)/2 \rceil$, pročež $d(v)$ je buď rovno $2d'(v)$ nebo o 1 nižší. Naučíme se rozpoznat, jestli $d(v)$ má být sudé nebo liché, a z toho vždy poznáme, jestli je potřeba jedničku odečíst.

Jak vypadá funkce d na sousedech vrcholu $v \neq i$? Pro alespoň jednoho souseda u je $d(u) = d(v) - 1$ (to platí pro sousedy, kteří leží na některé z nejkratších cest z v do i). Pro všechny ostatní sousedy je $d(u) = d(v)$ nebo $d(u) = d(v) + 1$.

Pokud je $d(v)$ sudé, vyjde pro sousedy ležící na nejkratších cestách $d'(u) = d'(v)$ a pro ostatní sousedy $d'(u) \geq d'(v)$, takže průměr z $d'(u)$ přes sousedy je alespoň $d'(v)$. Je-li naopak $d(v)$ liché, musí být pro sousedy na nejkratších cestách $d'(u) < d(v)$ a pro všechny ostatní $d'(u) = d(v)$, takže průměr klesne pod $d'(v)$.

Průměry přes sousedy přitom můžeme spočítat násobením matic: vynásobíme matici vzdáleností D' maticí sousednosti grafu G. Na pozici i, j se objeví součet hodnot D'_{ik} přes všechny sousedy k vrcholu j. Ten stačí vydělit stupněm vrcholu j a hledaný průměr je na světě.

Po provedení jednoho násobení matic tedy dovedeme pro každou dvojici vrcholů v konstantním čase spočítat D_{ij} z D'_{ij}. Jedna úroveň rekurze proto trvá $\mathcal{O}(n^\omega)$ a jelikož průměr grafu pokaždé klesne alespoň dvakrát, je úrovní $\mathcal{O}(\log n)$ a celý algoritmus doběhne ve slíbeném čase $\mathcal{O}(n^\omega \log n)$.

Literatura

[1] T. Chan. All-Pairs Shortest Paths with Real Weights in $O(n^3/\log n)$ Time. *Algorithmica*, 50(2):236–243, February 2008.

[2] H. Cohn, R. Kleinberg, B. Szegedy, and C. Umans. Group-theoretic Algorithms for Matrix Multiplication. *Foundations of Computer Science, Annual IEEE Symposium on*, pages 379–388, 2005.

[3] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of symbolic computation*, 9(3):251–280, 1990.

[4] Raz, R. On the complexity of matrix product. In *STOC '02: Proceedings of the 34th annual ACM Symposium on Theory of Computing*, pages 144–151, 2002.

[5] R. Seidel. On the all-pairs-shortest-path problem. In *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*, pages 745–749. ACM, 1992.

[6] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 13(4):354–356, 1969.

[7] Williams, V. V. Multiplying matrices faster than Coppersmith-Winograd. In *STOC '12: Proceedings of the 44th annual ACM Symposium on Theory of Computing*, pages 887–898, 2012.

[8] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM (JACM)*, 49(3):289–317, 2002.

[9] U. Zwick. A Slightly Improved Sub-Cubic Algorithm for the All Pairs Shortest Paths Problem with Real Edge Lengths. *Algorithmica*, 46(2):181–192, October 2006.