Programátorská Encyklopedie

KORESPONDENČNÍHO SEMINÁŘE Z PROGRAMOVÁNÍ

Hopcroft-Karpův algoritmus

Hopcroft-Karpův algoritmus slouží k rychlému nalezení maximálního párování v bipartitním grafu.[1] Vychází z podobné myšlenky jako základní algoritmus popsaný v článku o párování (postupné hledání zlepšujících cest).

Při odhadu složitosti tohoto algoritmu jsme vycházeli z toho, že při hledání zlepšující cesty projdeme v nejhorším případě až celý graf (což nám potrvá pro husté grafy až $\mathcal{O}(N^2)$), čímž toho o něm zjistíme hodně, ale nakonec z toho použijeme jedinou zlepšující cestu délky $\mathcal{O}(N)$ a všechny ostatní informace získané prohledáváním zahodíme. Nešlo by to lépe?

Tady použijeme jeden z populárních informatických triků: když se neumíme zbavit nějaké náročné operace, zkusme ji alespoň využít k tomu, abychom udělali co nejvíce užitečné práce.[2] Kdybychom předpokládali, že *pokaždé* projdeme graf celý, složitost jednoho hledání (říkejme mu *fáze*) se nezmění, ale zlepšujících cest najdeme potenciálně mnoho. Což v ideálním případě způsobí, že bude potřeba méně fází pro nalezení maximálního párování, a tedy bude celý algoritmus rychlejší.

Ovšem nalezené zlepšující cesty se mohou (a budou) překrývat a pokud nějakou použijeme ke zlepšení párování, spousta jiných může najednou přestat existovat nebo naopak vzniknout. Abychom tomuto předešli, budeme hledat *disjunktní* zlepšující cesty, tedy takové, které nesdílí žádné vrcholy. Pak určitě můžeme všechny nezávisle na sobě použít ke zvětšení párování o více než jedna za fázi. Tomuto postupu se říká *Hopcroft-Karpův algoritmus*.

Přesněji: v každé fázi najdeme nějakou v inkluzi maximální množinu[3] disjunktních nejkratších zlepšujících cest a každou z nich použijeme ke zlepšení párování.

Toto hledání se bude skládat ze dvou částí: BFS a DFS. V obou částech budeme provádět něco jako prohledávání grafu do šířky/hloubky, ale ne tak úplně: chceme chodit pouze po *potenciálně zlepšujících* cestách (tedy střídavých cestách začínajících volným vrcholem v levé partitě; o konci nic nevíme, dokud na něj nenarazíme).

Víme, že na takovýchto cestách půjdeme zleva doprava vždy po nepárovací hraně a zprava doleva po párovací. To nám umožní dívat se na to jako na prohledávání orientovaného grafu, kde nepárovací hrany jsou orientované zleva doprava a párovací zprava doleva. Potenciálně zlepšující cesty pak odpovídají právě orientovaným cestám z volných vrcholů levé partity. Pokud takováto orientovaná cesta navíc končí volným vrcholem, je zlepšující. Na algoritmu nic nezměníme, ale bude se nám o něm snáze přemýšlet, neb prohledávání orientovaného grafu je něco důvěrně známého.

Průběh algoritmu

V BFS části graf takto orientovaně projdeme do šířky ze všech volných vrcholů v levé partitě (musíme začít „ze všech současně“; jinými slovy prostě na začátku do fronty umístíme všechny volné levé vrcholy), čímž pro každý vrchol určíme délku nejkratší potenciálně zlepšující cesty do něj vedoucí.

Takto rozdělíme vrcholy grafu do pomyslných vrstev podle této vzdálenosti. Leč vzhledem k tomu, že nás zajímají pouze nejkratší zlepšující cesty, jakmile poprvé narazíme na konec zlepšující cesty, můžeme aktuální vrstvu prohlásit za poslední a po ní prohledávání ukončit – cokoli za ní už je moc daleko a nezajímá nás. Označíme-li ℓ délku nejkratší zlepšující cesty v grafu, rozdělili jsme vrcholy na vrstvy $0..\ell$ a hromadu nezařazených.

V DFS části procházíme graf do hloubky, opět dle orientace hran, opět ze všech volných vrcholů v levé partitě, ale tentokrát postupně. Při tom používáme BFS vrstvy jako vodítko – sestupujeme vždy pouze do vrcholů z následující vrstvy. Tím máme zaručeno, že všechny cesty, které najdeme, budou nejkratší. Pokaždé, když prohledávání dojde k volnému vrcholu vpravo, našli jsme zlepšující cestu a použijeme ji ke zvětšení párování (to si v naší analogii lze představit jako otočení orientace všech hran na cestě).

V tuto chvíli aktuální prohledávání ukončíme, neb vrchol, ze kterého jsme začínali, již není volný, a začneme znovu z následujícího volného vrcholu v levé partitě. DFS značky uchováváme napříč všemi vyhledáváními v dané fázi, abychom zbytečně pokaždé znovu nenavštěvovali podstromy, které k žádnému volnému vrcholu nevedou. Takto máme zajištěno, že každý vrchol a hranu navštívíme nejvýše jednou za fázi, a tedy celou DFS část (a BFS rovněž) zvládneme v $\mathcal{O}(|E|)$.

Časový odhad

Ukážeme, že takovémuto algoritmu stačí na nalezení maximálního párování $\mathcal{O}(\sqrt{|M|})$ fází. Nejprve si uvědomíme, že po každé fázi vzroste délka nejkratší zlepšující cesty alespoň o jedničku. Pokud na začátku fáze byla ℓ, pro každý volný levý vrchol, ze kterého vedla zlepšující cesta délky ℓ, jsme na této obrátili hrany, pročež řečený vrchol již není volný. Tedy všechny zlepšující cesty délky ℓ, které na začátku fáze existovaly, jsme zničili.

Žádné nové volné vrcholy jsme ani v jedné partitě nevytvořili. Ale nemohlo se stát, že jsme nějakou zlepšující cestu zkrátili? Není těžké dokázat (třeba indukcí podle vrstev), že otočením hran na nějaké nejkratší cestě nemůžeme žádnému vrcholu snížit vzdálenost. Tedy po k fázích bude nejkratší zlepšující cesta dlouhá alespoň $k+1$.

Pokračovat budeme podlým trikem: představíme si, že algoritmus zastavíme po $\sqrt{|M|}$ fázích a podíváme se na dosud nalezené párování P. Dále uvažujme libovolné maximální párování M a sestrojme „modro-purpurový“ rozdílový graf stejně jako v důkazu existence zlepšující cesty. A ze stejného rozboru možných komponent tohoto grafu vidíme, že $|M|-|P| = \#$ komponent tvořených cestou s oběma konci modrými (říkejme jim *hezké komponenty*).

Každá hezká komponenta odpovídá zlepšující cestě v P, tedy dle předchozího obsahuje alespoň $\sqrt{|M|}$ hran. Dále si všimneme, že celkem je v rozdílovém grafu nejvýše $2|M|$ hran. Tedy v grafu je nejvýše $2|M|/\sqrt{|M|}$ hezkých komponent, tedy $|M|-|P| = \mathcal{O}(\sqrt{|M|})$. A protože v každé fázi zvětšíme P alespoň o jedničku, do konce algoritmu zbývá nejvýše $\mathcal{O}(\sqrt{|M|})$ fází. Tedy i celkový počet fází je $\mathcal{O}(\sqrt{|M|})$.

Článek pro vás sepsal
Filip Štědronský

[1] `http://ksp.mff.cuni.cz/encyklopedie/parovani.html`

[2] Aneb účinnost lze získat nejen snižováním příkonu, ale také zvyšováním výkonu ;-).

[3] Tedy takovou, ke které se již nedá žádná další přidat (to nemusí nutně znamenat, že je největší možná).

Programátorská Encyklopedie

KORESPONDENČNÍHO SEMINÁŘE Z PROGRAMOVÁNÍ

Hopcroft-Karpův algoritmus

Hopcroft-Karpův algoritmus slouží k rychlému nalezení maximálního párování v bipartitním grafu.[1] Vychází z podobné myšlenky jako základní algoritmus popsaný v článku o párování (postupné hledání zlepšujících cest).

Při odhadu složitosti tohoto algoritmu jsme vycházeli z toho, že při hledání zlepšující cesty projdeme v nejhorším případě až celý graf (což nám potrvá pro husté grafy až $\mathcal{O}(N^2)$), čímž toho o něm zjistíme hodně, ale nakonec z toho použijeme jedinou zlepšující cestu délky $\mathcal{O}(N)$ a všechny ostatní informace získané prohledáváním zahodíme. Nešlo by to lépe?

Tady použijeme jeden z populárních informatických triků: když se neumíme zbavit nějaké náročné operace, zkusme ji alespoň využít k tomu, abychom udělali co nejvíce užitečné práce.[2] Kdybychom předpokládali, že *pokaždé* projdeme graf celý, složitost jednoho hledání (říkejme mu *fáze*) se nezmění, ale zlepšujících cest najdeme potenciálně mnoho. Což v ideálním případě způsobí, že bude potřeba méně fází pro nalezení maximálního párování, a tedy bude celý algoritmus rychlejší.

Ovšem nalezené zlepšující cesty se mohou (a budou) překrývat a pokud nějakou použijeme ke zlepšení párování, spousta jiných může najednou přestat existovat nebo naopak vzniknout. Abychom tomuto předešli, budeme hledat *disjunktní* zlepšující cesty, tedy takové, které nesdílí žádné vrcholy. Pak určitě můžeme všechny nezávisle na sobě použít ke zvětšení párování o více než jedna za fázi. Tomuto postupu se říká *Hopcroft-Karpův algoritmus*.

Přesněji: v každé fázi najdeme nějakou v inkluzi maximální množinu[3] disjunktních nejkratších zlepšujících cest a každou z nich použijeme ke zlepšení párování.

Toto hledání se bude skládat ze dvou částí: BFS a DFS. V obou částech budeme provádět něco jako prohledávání grafu do šířky/hloubky, ale ne tak úplně: chceme chodit pouze po *potenciálně zlepšujících* cestách (tedy střídavých cestách začínajících volným vrcholem v levé partitě; o konci nic nevíme, dokud na něj nenarazíme).

Víme, že na takovýchto cestách půjdeme zleva doprava vždy po nepárovací hraně a zprava doleva po párovací. To nám umožní dívat se na to jako na prohledávání orientovaného grafu, kde nepárovací hrany jsou orientované zleva doprava a párovací zprava doleva. Potenciálně zlepšující cesty pak odpovídají právě orientovaným cestám z volných vrcholů levé partity. Pokud takováto orientovaná cesta navíc končí volným vrcholem, je zlepšující. Na algoritmu nic nezměníme, ale bude se nám o něm snáze přemýšlet, neb prohledávání orientovaného grafu je něco důvěrně známého.

Průběh algoritmu

V BFS části graf takto orientovaně projdeme do šířky ze všech volných vrcholů v levé partitě (musíme začít „ze všech současně“; jinými slovy prostě na začátku do fronty umístíme všechny volné levé vrcholy), čímž pro každý vrchol určíme délku nejkratší potenciálně zlepšující cesty do něj vedoucí.

Takto rozdělíme vrcholy grafu do pomyslných vrstev podle této vzdálenosti. Leč vzhledem k tomu, že nás zajímají pouze nejkratší zlepšující cesty, jakmile poprvé narazíme na konec zlepšující cesty, můžeme aktuální vrstvu prohlásit za poslední a po ní prohledávání ukončit – cokoli za ní už je moc daleko a nezajímá nás. Označíme-li ℓ délku nejkratší zlepšující cesty v grafu, rozdělili jsme vrcholy na vrstvy $0..\ell$ a hromadu nezařazených.

V DFS části procházíme graf do hloubky, opět dle orientace hran, opět ze všech volných vrcholů v levé partitě, ale tentokrát postupně. Při tom používáme BFS vrstvy jako vodítko – sestupujeme vždy pouze do vrcholů z následující vrstvy. Tím máme zaručeno, že všechny cesty, které najdeme, budou nejkratší. Pokaždé, když prohledávání dojde k volnému vrcholu vpravo, našli jsme zlepšující cestu a použijeme ji ke zvětšení párování (to si v naší analogii lze představit jako otočení orientace všech hran na cestě).

V tuto chvíli aktuální prohledávání ukončíme, neb vrchol, ze kterého jsme začínali, již není volný, a začneme znovu z následujícího volného vrcholu v levé partitě. DFS značky uchováváme napříč všemi vyhledáváními v dané fázi, abychom zbytečně pokaždé znovu nenavštěvovali podstromy, které k žádnému volnému vrcholu nevedou. Takto máme zajištěno, že každý vrchol a hranu navštívíme nejvýše jednou za fázi, a tedy celou DFS část (a BFS rovněž) zvládneme v $\mathcal{O}(|E|)$.

Časový odhad

Ukážeme, že takovémuto algoritmu stačí na nalezení maximálního párování $\mathcal{O}(\sqrt{|M|})$ fází. Nejprve si uvědomíme, že po každé fázi vzroste délka nejkratší zlepšující cesty alespoň o jedničku. Pokud na začátku fáze byla ℓ, pro každý volný levý vrchol, ze kterého vedla zlepšující cesta délky ℓ, jsme na této obrátili hrany, pročež řečený vrchol již není volný. Tedy všechny zlepšující cesty délky ℓ, které na začátku fáze existovaly, jsme zničili.

Žádné nové volné vrcholy jsme ani v jedné partitě nevytvořili. Ale nemohlo se stát, že jsme nějakou zlepšující cestu zkrátili? Není těžké dokázat (třeba indukcí podle vrstev), že otočením hran na nějaké nejkratší cestě nemůžeme žádnému vrcholu snížit vzdálenost. Tedy po k fázích bude nejkratší zlepšující cesta dlouhá alespoň $k+1$.

Pokračovat budeme podlým trikem: představíme si, že algoritmus zastavíme po $\sqrt{|M|}$ fázích a podíváme se na dosud nalezené párování P. Dále uvažujme libovolné maximální párování M a sestrojme „modro-purpurový“ rozdílový graf stejně jako v důkazu existence zlepšující cesty. A ze stejného rozboru možných komponent tohoto grafu vidíme, že $|M|-|P| = \#$ komponent tvořených cestou s oběma konci modrými (říkejme jim *hezké komponenty*).

Každá hezká komponenta odpovídá zlepšující cestě v P, tedy dle předchozího obsahuje alespoň $\sqrt{|M|}$ hran. Dále si všimneme, že celkem je v rozdílovém grafu nejvýše $2|M|$ hran. Tedy v grafu je nejvýše $2|M|/\sqrt{|M|}$ hezkých komponent, tedy $|M|-|P| = \mathcal{O}(\sqrt{|M|})$. A protože v každé fázi zvětšíme P alespoň o jedničku, do konce algoritmu zbývá nejvýše $\mathcal{O}(\sqrt{|M|})$ fází. Tedy i celkový počet fází je $\mathcal{O}(\sqrt{|M|})$.

Článek pro vás sepsal
Filip Štědronský

[1] `http://ksp.mff.cuni.cz/encyklopedie/parovani.html`

[2] Aneb účinnost lze získat nejen snižováním příkonu, ale také zvyšováním výkonu ;-).

[3] Tedy takovou, ke které se již nedá žádná další přidat (to nemusí nutně znamenat, že je největší možná).