Korespondenční Seminář z Programování

38. ročník **KSP** **Prosinec 2025**

Vzorová řešení druhé série třicátého osmého ročníku KSP

38-2-1 Všechny cesty vedou přes Řím

První si označme Kolín jako vrchol u a Brno jako vrchol v. Teď pojďme přemýšlet v řeči grafů. Vrchol na všech uv-cestách (všech cestách z u do v) si označme dominující. Cíl úlohy je pro každý vrchol zjistit, zda je dominující.

Ve zbytku řešení zafixujeme nějakou libovolnou uv-cestu, označme ji P. Vrcholy $x \notin P$ z definice nejsou dominující (jelikož neleží na všech uv-cestách) a vrchol $x \in P$ není dominující, právě když jde najít nějaká cesta P' taková, že $x \notin P'$. Intuitivně se pro každé $x \in P$ snažíme zjistit, zda jde "obejít".

Jak může taková P' vypadat? Cest s $x \notin P'$ může být opravdu hodně, ale dá se rozmyslet, že pokud x není dominující, tak vždycky bude existovat cesta P', která začíná stejně jako P, pak jde nějakou dobu výhradně po hranách a vrcholech mimo P a pak končí stejně jako P.

Označme vrcholy na cestě P popořadě jako $w_1, w_2, \ldots, w_\ell$. Předchozí pozorování se dá zformalizovat do následujícího tvrzení: vrchol $w_j \in P$ není dominující, právě když existují vrcholy w_i a w_k pro $i < j < k$ takové, že mezi w_i a w_k se dá dojít po vrcholech mimo P. Jeho pravdivost nahlédneme snadno: pokud takové dva vrcholy existují, pak vrchol w_j můžeme obejít. Naopak pokud w_j můžeme obejít, tj. pokud existuje $P' \not\ni w_j$, tak se v P' dá najít vhodné w_i a w_k.

Stačí nám tedy pro každý vrchol w_i spočítat, do jakého nejpravějšího vrcholu $w_k \in P$ z něj umíme najít obchůzku. Označme index takového vrcholu jako k_i.

Spočítat k_i můžeme například prohledáváním do hloubky upraveným tak, že si zakážeme používat vrcholy v P. Průběžně si udržujeme nejpravější vrchol P, který jsme dosud viděli, a na konci jeho index uložíme jako k_i.

Naivní řešení spočítá k_i pro všechna w_i v čase $\mathcal{O}(|P|(M+N)) \subset \mathcal{O}(N(M+N))$. Teď máme $|P|$ intervalových tvrzení tvaru "vrcholy mezi i a k_i nejsou dominující". Naivně pak můžeme v čase $\mathcal{O}(|P|^2) \subset \mathcal{O}(N^2)$ pro každý vrchol rozhodnout, zda je dominující. Celková časová složitost je $\mathcal{O}(N(M+N))$.

Jak tento algoritmus zrychlit? Důležité je následující pozorování: vůbec nemusíme počítat k_i, stačí nám jen vždycky určit, jestli se z w_i umíme dostat dál než ze všech předchozích w_j pro $j < i$, a případně kam. Pokud ne, tak nám vrchol w_i nedává žádnou novou informaci (obchůzka z něj je ostře horší než nějaká jiná). Speciálně při DFS z w_i stačí navštěvovat jen vrcholy, které jsme ještě nenavštívili v předchozích voláních: u těch navštívených už totiž víme, že nám nepomohou objevit z w_i lepší obchůzku. Můžeme tedy všechna DFS pustit s jednou globální značkou "navštíveno" pro každý vrchol, kterou mezi voláními neresetujeme. Tak za celý běh algoritmu navštívíme každý vrchol a hranu nejvýše jednou a celková časová složitost prohledávání bude jen $\mathcal{O}(N+M)$.

S takto upraveným algoritmem umíme snadno zjistit, které vrcholy jsou dominující. Udržujeme si m, index nejvzdálenějšího vrcholu na P, do kterého se umíme dostat z nějakého $j < i$. Pokud při příchodu do vrcholu w_i platí $m \leq i$, pak je vrchol w_i dominující, jinak není.

Celkem nám algoritmus zabere času v $\mathcal{O}(N+M)$ pro nalezení P a dalších $\mathcal{O}(N+M)$ pro nalezení obejitelných vrcholů na P.

Úlohu připravili: Ríša Hladík, Katia „Čiči" Kočická, Kristýna Petrlíková, Matúš Púll

38-2-2 Stop STOPkám

Zhrnutie

Úloha od nás chce aby sme našli takú cestu z vrcholu 1 do vrcholu N, ktorá prioritne minimalizuje počet zastavení na STOPkách a sekundárne dĺžku cesty. Je známe, že ak by sme chceli iba minimalizovať dĺžku cesty, tak by sme mohli použiť napríklad Dijkstrov algoritmus. V tejto úlohe však máme dve kritériá, ktoré chceme minimalizovať, avšak ukážeme si že po malej modifikácii to Dijkstrovým algritmom stále pôjde.

Dijkstrov algoritmus

Dijkstrov algoritmus funguje tak, že postupne hľadá najkratšie cesty do všetkých vrcholov zo začiatočného vrcholu. Začne v začiatočnom vrchole, do ktorého sa vieme dostať prejdením nulovej vzdialenosti. Následne pre každého jeho suseda vieme že sa do neho vieme dostať prejdením vzdialenosti rovnajúcej sa vzdialenosti do aktuálneho vrcholu plus dĺžke hrany medzi nimi. V ďalšom kroku vyberieme vrchol s najmenšou vzdialenosťou, ktorú sme zatiaľ našli, a ešte sme z neho neprechádzali jeho susedov, a opakujeme proces, teda zisťujeme najkratšie cesty do jeho susedov. Detaily o Dijkstrovom algoritme môžete nájsť na Wiki stránke.[1]

Modifikácia Dijkstrovho algoritmu

Použitie dvoch kritérií nám úlohu mierne komplikuje, avšak vieme si všimnúť, že algoritmus vie fungovať aj s viacerími kritériami. Stačí, aby sme namiesto ukladania jednej hodnoty vzdialenosti ukladali dvojicu hodnôt, počet zastavení a vzdialenosť. Porovnanie bude fungovať lexikograficky, teda najprv porovnáme počet zastavení a ak sú rovnaké, tak porovnáme vzdialenosť. Vidíme že takéto porovnanie vždy prioritne minimalizuje počet zastavení a sekundárne dĺžku cesty, čo je presne to, čo chceme. Teda po tejto modifikácií vieme spustiť štandardný Dijkstrov algoritmus.

Ako riešiť hlavné a vedľajšie cesty?

Implementačný trik na či je cesta je hlavná alebo vedľajšia je, že si v každom vrchole zoradíme hrany podľa priority, a následne sa vždy iba pozrieme či je hrana po ktorej sme prišli medzi dvoma hranami s najvyššou prioritou.

Ak nie je, tak sme prišli po vedľajšej ceste a musíme pripočítať jedno zastavenie, pretože budeme musieť čakať na STOPke. Je si avšak treba dať pozor na špeciálny prípad, keď prídeme do koncového vrcholu N, tam totiž čakať nemusíme, takže vtedy zastavenie nepripočítavame.

[1] `https://cs.wikipedia.org/wiki/Dijkstr%C5%AFv_algoritmus`

Ako rekonštruovať cestu?

Cestu vieme rekonštruovať štandardným spôsobom, teda si pri každom vrchole pamätáme takzvaný back-pointer, teda z ktorého vrcholu sme do neho prišli po najkrašej ceste. Po skončení algoritmu vieme začať v koncovom vrchole N a postupne chodiť po back-pointeroch až kým nedôjdeme do vrcholu 1. Týmto spôsobom získame cestu v opačnom poradí, takže ju ešte treba otočiť.

Zložitost

Dijkstrov algoritmus s použitím prioritnej fronty beží v čase $\mathcal{O}((M+N)\log N)$, kde N je počet vrcholov a M počet hrán.

Pamäťová zložitosť je $\mathcal{O}(N+M)$ na uloženie grafu a ďalších $\mathcal{O}(N)$ na ukladanie vzdialeností a back-pointerov.

Program (C++):
`http://ksp.mff.cuni.cz/viz/38-2-2.cpp`

Úlohu připravili: Patrik Prítrský, Dan Skýpala

38-2-3 Opisování

Pojďme si nejdřív rozmyslet, s čím to vlastně pracujeme: Dostali jsme tabulku $N \times M$, tedy víme pro každého studenta, kterou úlohu umí. Co ale pro nás bude důležitější, je, že víme pro každou úlohu, který student ji umí – a tedy pro každou úlohu, kolik studentů ji umí.

Pro každou úlohu si tedy spočítáme, kolik studentů ji umí. Teď vezměme tu, kterou umí co nejméně studentů – neboli kterou neumí co nejvíce studentů. Když před neznalými studenty nebude žádný znalý, tak všechny neznalé s jistotou vyhodíme!

Jde to ale lépe? Můžeme teď zase třídit sekundárně podle druhé nejméně známé otázky, a tak dále? Myšlenka je to lákavá, ale ukážeme si, že ne: nechť K je počet studentů, kteří neumí nejtěžší otázku. Platí, že ať uspořádáme studenty jakkoliv, $(K+1)$-tý z nich (a tedy i každý další) správně odpoví na všechny otázky. To proto, že pro každou otázku existuje nanejvýš K studentů, kteří ji neumějí, tudíž mezi $K+1$ prvními studenty bude aspoň jeden, který tuto otázku umí. $(K+1)$-tý student tedy díky opisování správně odpoví všechny otázky.

Naše řešení jednou projde celou tabulku $N \times M$, pro každou otázku zjistí počet znalých studentů a nakonec všechny znalé studenty nasází nakonec a neznalé dopředu, což zvládneme v čase $\Theta(N)$. Celková časová složitost bude tedy v $\Theta(NM)$.

Úlohu připravili: Matúš Púll, Ben Swart

38-2-4 Zákeřná parkovací místa

S dovolením budeme referovat pouze k hodnotám parkovacích míst a jejích umístění v řadě. To, že jsou to parkovací místa a že na nich stojí auta, ačkoliv nás naplňuje nepokojem, protože jistě alespoň polovina ze zúčastněných mohla přijet městskou hromadnou dopravou a ušetřit tím jak peníze, tak životní prostředí, není algoritmicky zajímavé.

Co vlastně můžeme usoudit, když se podíváme na hodnotu na jednom místě? A ono vlastně vůbec nic – samotná hodnota jednoho místa nám nic neřekne. Když se ale podíváme i na jeho sousedy, hned z toho můžeme něco zjistit. Mohou nastat tři situace:

- Oba sousedi mají nižší/stejnou hodnotu – našli jsme hledané místo.
- Jeden soused má nižší/stejnou hodnotu a jeden vyšší – co v tu chvíli můžeme usoudit o směru, ve kterém je ten vyšší? Jelikož hodnoty nemůžou růst pořád, páč by jistě narazily na okraji, tak se někde tímto směrem určitě nachází nějaké vyhovující místo. Tedy se můžeme koukat jenom na místa tímto směrem s jistotou, že tam nějaké vyhovující místo je.
- Oba sousedi mají vyšší hodnotu – můžeme si vybrat, kterým směrem se vydáme a aplikujeme předchozí tvrzení.

No a nakonec už si akorát všimneme, že takhle se můžeme podívat do poloviny prohledávaného úseku a tím polovinu jistě zahodit. Tedy v každém kroku snížíme počet prohledávaných pozic na zhruba polovinu. Tedy celkem uvidíme $3 \cdot \log_2 N$ pozic, což je v $\mathcal{O}(\log N)$.

Poznámka na závěr: pokud bychom chtěli zlepšovat konstantu před $\log_2 N$, existuje na to chytrý algoritmus jménem metoda zlatého řezu. Ta používá podobnou myšlenku, že ze tří hodnot jsme schopni vyloučit část parkoviště, ale místo tří sousedních parkovacích míst volí místa trochu složitěji, čímž na každé úrovni rekurze musíme vyhodnotit jen jednu novou pozici. Nevýhodou je, že rekurze je trochu hlubší, jelikož pole nepůlíme přesně v půlce. Přesto počet dotazů díky tomu klesne, a to konkrétně na zhruba $1{,}44 \log_2 N$. Pokud jste zvědaví, můžete si přečíst víc na anglické Wikipedii.[2]

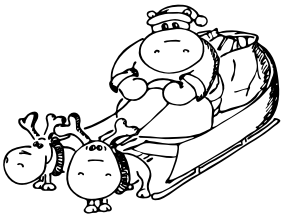
Úlohu připravili: Daniel Culliver, Kristýna Petrlíková, Matúš Púll

38-2-S Hledá se pivot

Řešení seriálu najdete na našem webu:

`http://ksp.mff.cuni.cz/viz/38-2/reseni`

Úlohu připravili: Martin „Medvěd“ Mareš, Dan Skýpala



[2] `https://en.wikipedia.org/wiki/Golden-section_search`

KSP pro vás připravují studenti Matematicko-fyzikální fakulty Univerzity Karlovy.
Realizace projektu byla podpořena Ministerstvem školství, mládeže a tělovýchovy.

Webové stránky: *https://ksp.mff.cuni.cz/*
E-mail: *ksp@mff.cuni.cz*
Organizátoři a kontakty: *https://ksp.mff.cuni.cz/kontakty/*