Příprava na státnicový okruh Grafové algoritmy

Filip Kastl

3. června 2026

Verze 1
Verze 2: Minimální kostry: Cleanup a pozn., že algoritmy z GA2 netřeba umět.
Verze 3: LCA: Oprava časové složitosti makrostuktury. Karger-Stein: Myslím, že vím jak implementovat vážené hrany.
Verze 4: Karger-Stein: Fix typo. Tranzitivní uzávěr: Doplněn rozděl a panuj algoritmus.
Verze 5: Minimální kostry: Kontr. borůvka z paralelních hran nechá vždy *nejlehčí*, ne nejtěžší. Nadpis „O následujících algoritmech stačí jen vědět“ *rightarrow* „Stačí jen vědět, že následující alg. existují“. Fredman-Tarjan je lin. pro $\exists k : m \in \Omega(n \log^{(k)} n)$. Union-Find: Má složitost $\mathcal{O}(\alpha(m, n))$, ne $\mathcal{O}(\alpha(n, n))$, odebráno TODO.

Tento dokument jsem sepsal zatímco jsem se učil na státnicový okruh Grafové algoritmy. Snažím se zde shrnout všechnu látku z magisterských Diskrétních modelů i bakalářské Obecné informatiky, s kterou jsem se potkal a myslím si, že spadá do tohoto okruhu. Je toho opravdu hodně. Už jsem výběr trochu prořezal – odsunul jsem některé věci do druhé sekce. Pravděpodobně se ale ani všechno z první sekce nebudu učit. Každopádně doufám, že tento výčet poslouží mně i ostatním jako dobrý startovní bod při učení se na státnice. Hodně štěstí!

1 O čem se dá mluvit

1.1 Pokročilé algoritmy pro nejkratší cesty

Pozn.: *V tomhle podtématu předpokládáme no negative cycles.*

Algoritmus: Bellman-Ford-Moore $\mathcal{O}(nm)$*, ale umí se vypořádat s negativními hranami*

Věta: *Dijkstra nikdy znovu neotevře uzavřený vrchol a zavírá vrcholy v pořadí, kde neklesá* $d(s, v)$.

Pozn.: *Dijkstra udělá* $\mathcal{O}(n)\times$ `Insert`*,* $\mathcal{O}(n)\times$ `ExtractMin` *a* $\mathcal{O}(m)\times$ `Decrease`

DS	`Insert`	`ExtractMin`	`Decrease`	Dijkstra
Pole	1	n	1	$\mathcal{O}(n^2)$
Bin. halda	$\log n$	$\log n$	$\log n$	$\mathcal{O}((m+n)\log n)$
d-reg. halda	$\mathcal{O}(\log n / \log d)$	$\mathcal{O}(d \cdot \log n / \log d)$	$\mathcal{O}(\log n / \log d)$	$\mathcal{O}((m+dn)\log n / \log d)$
$\frac{m}{n}$-reg. halda				$\mathcal{O}(m \log n / \log \frac{m}{n})$
Fibbonacci	$\mathcal{O}(1)$	$\mathcal{O}(\log n)$	$\mathcal{O}(1)$	$\mathcal{O}(m + n \log n)$
Délky $\leq L$	$\mathcal{O}(1)$	$\mathcal{O}(\log L / \log \log L)$	$\mathcal{O}(1)$	$\mathcal{O}\left(m + n \cdot \frac{\log L}{\log \log L}\right)$

Pozn.: *Poslední halda pracuje na celých číslech. Dá se rozšířit i na neceločíselné hodnoty, ale pak musím uvažovat* $L' := L/\lambda, \forall e : \lambda \leq l(e) \leq L$

Pozn.: *Na* $d := \frac{m}{n}$ *jsme přišli tak, že jsme chtěli, aby oba členy* ($m \cdot$ *zlomek* $+ n \cdot$ *zlomek*) *byly stejné (to je asymptoticky opt.).*

Efektivně se složitost pak volně pohybuje mezi polem a bin. haldou. Medvěd říká, že v praxi je to lepší, než Fibonacciho halda (v té se prý skrývají ošklivé konstanty).

Pozn.: *Relaxační algoritmy jako Dijkstra a Bellman-Ford řeší single-source-shortest-paths. Medvěd: Vlastně se neví, jak využít toho, že reálně chceme řešit point-to-point-shortest-path k vylepšení asymptotiky. Ale máme heuristiky!*

Heuristika: *Zastav, když uzavřeš cílový vrchol*

Heuristika: *Prohledávej ze zdroje i z cíle najednou (je třeba dokázat)*

Definice: Potenciál $\varphi : V \to \mathbb{R}_0^+$

Definice: Reduced length $l_\varphi(uv) := l(uv) + \varphi(u) - \varphi(v)$

Pozorování: *Reduced délka cesty* P *se zteleskopuje na* $l(P) + \varphi(u) - \varphi(v)$. *Tedy nejkratší cesta je nejkratší i po redukování délek.*

Definice: Feasible potenciál $\forall e : l_\varphi(e) \geq 0$

Věta: *Pokud no negative cycles, existuje feasible potenciál (pomocí extra vrcholu a BFM)*

Pozn.: *S tímhle potenciálem lze pustit Dijkstru i na grafy s negativními hranami.*

Algoritmus: A* $\psi(v) :=$ *dolní odhad na* $d(v, t)$. *Zavíráme vždy vrchol s nejmenším* $h(v) + \psi(v)$ ($h(v)$ *je délka dosud nalezené nejkratší cesty do* v).

Věta: *A* je ekvivalentní s Dijkstrou s potenciálem* $-\psi$

1.2 Tranzitivní uzávěr

Pozn.: *Zařazuji sem i all-pairs-shortest-paths, protože ty dva problémy úzce souvisí.*

Pozorování: *Taky bychom mohli prostě* n*-krát iterovat a dostat z Fibonacciho Dijkstry* $\mathcal{O}(mn + n^2 \log n)$ *APSP (ale FW je lehčí na implementaci) a z BFSka* $\mathcal{O}(mn)$ *tranz. uzávěr.*

Algoritmus: Floyd-Warshal $D^k_{i,j} := i, j$ *vzdálenost pokud povolíme jen* $v_i, v_j, v_1, \ldots, v_k$
$\mathcal{O}(n^3)$ *time,* $\mathcal{O}(n^2)$ *space.*

Pozn.: *FW se dá modifikovat, aby nejen našel vzdálenosti, ale také našel ty cesty.*

Pozn.: *Floyd-Warshal se dá použít pro řešení jiných problémů. Např. nejširší cesta, nejpravděpodobnější cesta, konstrukce gramatiky pro daný konečný automat.*

Trik je v tom nahradit operace. Třeba u nejširší cesty má neexistující walk hodnotu $-\infty$, *prázdný walk hodnotu* $+\infty$, *walk o jedné hraně prostě šířku té hrany, množina dvou walků maximum šířek těch walků, konkatenace dvou walků minimum šířek.*

Pokud bychom chtěli být fancy, uchopili bychom to jako homomorfismy z nějaké obecné algebry walků. Tak se to kinda vykládá na GA1. Ale to jde pro účely státnic asi moc do hloubky.

Definice: $(\oplus, \otimes)$**-produkt matic.** *Nechť* $\oplus$ *je komutativní a asociativní a* $\otimes$ *je asociativní.*

$$C_{ij} := \bigoplus_k A_{ik} \otimes B_{kj}$$

Algoritmus: $(\vee, \wedge)$*-produkt matic dává* $\mathcal{O}(n^\omega \log n)$ *tranz. uzávěr.*

Pozn.: $(\min, +)$-*produkt matic by dával APSP vzdálenosti, ale protože* min *nemá inverz, nelze použít ty nejchytřejší algoritmy násobení matic, a tak zůstáváme na* $\mathcal{O}(n^3/\log n)$ *násobení, s čímž FW nepřekonáme.*

Algoritmus: Rozděl a panuj zrychlení tranz. uzávěru

- $\mathcal{O}(n^\omega)$
- *Rozdělíme vstupní matici sousednosti a výstupní matici dosažitelnosti na čtyři podmatice a ukážeme, že je vlastně potřeba spočítat na každé úrovni rekurze jen dva podproblémy, ne čtyři.*
- *Pak dominantní je kořen stromu rekurze, kde nejvíce času zabere násobení (konstantně mnoha) matic* $\rightarrow \mathcal{O}(n^\omega)$.

Myšlenky algoritmu: Seidelův algoritmus $\mathcal{O}(n^\omega \log n)$ *full APSP, ale jen pro unweighted*

1.3 Toky v sítích

Algoritmus: Tokové z bakaláře

- *Ford-Fulkerson* $\mathcal{O}(nm^2)$ *(ale musíme hledat nejkratší (neváženou) cestu, také se tomu říká Edmonds-Karpův algoritmus)*
- *Dinic* $\mathcal{O}(n^2m)$
- *Goldberg* $\mathcal{O}(n^2m)$*, asi stačí znát jenom základní princip. Hvězdou je tady Dinic.*

Algoritmus: Dinic s Link/cut stromy $O(nm \cdot \log n)$ *Prý rychlé i v praxi.*

Analýza: Dinic na jednotkových vahách

- $\mathcal{O}(m^{3/2})$
- $\Theta(\sqrt{n} \cdot m)$ *pro* $\min(\deg^{in}(v), \deg^{out}(v)) \leq 1$
- $\Theta(n^{2/3} \cdot m)$ *pro non-multigrafy*

(pro ty Θ *jsme si ukazovali jenom* $\mathcal{O}$ *důkaz)*

Algoritmus: Varianty Dinice pro celočíselné kapacity $\leq C$

- $\mathcal{O}(nm + n^2C)$
- $\mathcal{O}(nm \cdot \log C)$

Pozn.: *Neprobrané algoritmy:*

- *3 indians (varianta Dinice)* $\mathcal{O}(n^3)$
- *Orlin* $\mathcal{O}(nm)$ *(to překonává 3 indy, ty celočíselné algoritmy a link-cut trees, ne? leda že oni jsou praktičtější)*
- *V roce 2022* $\mathcal{O}(m^{1+\epsilon}), \forall \epsilon \geq 0$

1.4 Řezy

Pozn.: *Pozor na rozdíl mezi libovolným (myslím, že se mu říká* globální*) řezem a st-řezem.*

U globálního řezu uvažujeme neorientovaný nevážený graf (Ale Karger-Stein lze upravit pro vážený graf – složitost není závislá na počtu hran, tak prostě použij multihrany, popř. reálná čísla jako počty hran. Myslím, že algoritmus s tímhle v poho bude fungovat.)

Věta: *Velikost maximálního toku = velikost minimálního st řezu.*

Algoritmus: *Globální řez pomocí tokového algoritmu najdu tak, že si zafixuji jeden vrchol s a pak najdu minimální st řez pro každé t.*

Algoritmus: *Globální separátor (vrcholová souvislost) pomocí tokového algoritmu: Rozdělím si každý vrchol na dva. Ale teď jsem si mohl jako s zafixovat jeden z vrcholů separátoru! Takže zkouším různé volby s, dokud nenajdu separátor menší než počet s, které jsem dosud vyzkoušel.*

Algoritmus: Contract

1. *Dokud $n > l$:*
2. *Vybereneme hranu $e \in E$ rovnoměrně náhodně*
3. *Zkontrahujeme hranu e, smyčky odstraníme, para. hrany ponecháme*

- *Řez pak najdeme bruteforcem*
- *Celou věc iterujeme – tomu Wikipedie říká „Kargerův algoritmus“.*

Myšlenky analýzy: Karger

- *Řez nezničíme s pvd. c/n^2, kde c je závislé na l. S iterováním a $e^{-x} \geq 1 - x$ máme pvd. úspěchu $1 - e^{-cK/n^2}$. Pro $K \in \Omega(n^2 \log n)$ stlačíme pvd. neúspěchu pod libovolný polynom.*
- *Graf reprezentujeme maticí sousednosti, kde prvky jsou počty paralelních hran.*
- *Kontrakce běží v $\mathcal{O}(n)$, zkontrahování až do l zvládneme v $\mathcal{O}(n^2)$, takže celkem $\mathcal{O}(n^4 \log n)$ pro inverzně polymiální pvd. neúspěchu.*

Algoritmus: MinCut

1. *Pokud $n \leq 7$, najdeme řez hrubou silou*
2. *$l := \lceil n/\sqrt{2} + 1 \rceil$*
3. *$C_1 := MinCut(Contract(G, l))$*
4. *$C_2 := MinCut(Contract(G, l))$*
5. *Vrátíme menší z C_1, C_2*

- *Celý Karger-Stein je pak iterování `MinCut`.*

Myšlenky analýzy: Karger-Stein

- *Problém se zmenšuje o $\sqrt{2}$, takže $\mathcal{O}(\log n)$ vrstev rekurze*
- *Nakonec se dobereme k $\mathcal{O}(n^2 \log n)$ runtimu `MinCut`*
- *Počítáme pvd. úspěchu na podproblému na i-té vrstvě jako rekurenci.*
- *Celkově $\mathcal{O}(n^2 \log^3 n)$ pro inverzně poly. pvd. neúspěchu.*

1.5 Link/cut stromy

Datová struktura: Heavy-light dekompozice (HLD)

- *Máme váhy buď na hranách nebo na vrcholech*
- *Queries přes cesty (např. minimum)*
- *Bodové (hrana/vrchol) a cestové updaty (např. „přičti δ všude po cestě“)*
- Těžká hrana *je ta, na které visí víc jak půlka podstromu.*
- *Základní vlastnosti:*
  1. *Z každého vrcholu dolů vede nejvýše jedna těžká hrana.*
  2. *Každý vrchol náleží do přesně jedné těžké cesty.*
  3. *Každá cesta z kořenu do listu obsahuje nejvýše* $\log n$ *lehkých hran.*
- *Intervalové stromy nad heavy paths* $\rightarrow$ *path ops v* $\mathcal{O}(\log^2 n)$ *worst-case (ale tohle Link/Cut stromy překonají).*
- *Pokud nechceme umět updatovat ani strukturu ani váhy, můžeme si navíc k intervalovým stromům předpočítat suffixy a odpovídat v* $\mathcal{O}(\log n)$.

Datová struktura: Link/cut stromy

- *Dynamická varianta HLD (tedy teď uvažujeme les).*
- *Interface:* `Parent(v)`, `Root(v)`, `Cut(v)`, `Link(u, v)`, `Cost(v)`, `PathCost(u, v)`, `SetCost(v)`, `PathUpdate(v, delta)`, `Evert(v)` *(ten udělá z v kořen)*
- *Jakmile přejdeme na link/cut stromy z HLD, časové složitosti jsou amortizované (existuje i worst-case verze, ale Medvěd ji neukazoval) –* $\mathcal{O}(\log n)$
- *A taky už hranám neříkáme těžké a lehké, ale tlusté a tenké.*
- *A taky už neplatí vlastnost 3.*
- *A taky používáme splay stromy místo intervalových stromů.*
- *Interface operace využívají interní operaci* `Expose(v)` *– z v udělá první (nejnižší) vrchol tlusté cesty do kořene.*

Myšlenky analýzy: Operace na Link/cut stromech v amort. $O(\log n)$ *Analýza je založená na analýze splay stromů.* $s(v)$ *je tentokrát počet descendants včetně těch dosažitelných tenkými hranami.* $r(v) := \log s(v)$ *jako u splay stromů. Naučil bych se asi akorát tolik, aby mi dávalo smysl, jak jsou implementované operace.*

Algoritmus: Dinic s Link/cut stromy $O(nm \cdot \log n)$ *Prý rychlé i v praxi.*

Zdroje

`Lecture Notes on Data Structures extra chapter "50. Graph data structures"`
`https://kam.mff.cuni.cz/~mares/video/ls2324/ds2/07-linkcut.mp4`
`https://kam.mff.cuni.cz/~mares/video/ls2324/ds2/08-linkcut2-semigroup.mp4`
`https://kam.mff.cuni.cz/~mares/video/ls2526/ds2/06-linkcut.mp4`

1.6 E-T stromy, plně dynamické udržování komponent souvislosti

Pozn.: *(a,b)-stromy mají $\mathcal{O}\left(\frac{\log n}{\log a}\right)$ `Find` a $\mathcal{O}\left(b \cdot \frac{\log n}{\log a}\right)$ `Insert` a `Delete`.*

Datová struktura: E-T stromy

- *Ukládá Eulerian-Tour sekvence jako listy (a,b)-stromů*
- *Nabízí `Cut`, `Reroot`, `Link`, `FindRoot`, vše v $\mathcal{O}(1)$ operacích na (a,b)-stromě*
- *Tyhle operace nám přímočaře dávají dynamickou konektivitu na lese.*

Pozn.: *Zatímco Link-Cut stromy nám dávají efektivní operace na cestách, E-T stromy nám dávají efektivní operace na podstromech. Např. některé vrcholy původního stromu obarvíme černě. Info, že vrchol je černý, si udržujeme v aktivních externích nodech. Interní nody si pamatují počet černých vrcholů pod sebou. Pokud chceme vědět, kolik černých vrcholů je v podstromu, můžeme ho odříznout a pak se podívat do kořene. Tohle celé je dynamické. Můžeme měnit barvu vrcholů a můžeme přidávat a ubírat hrany.*

Datová struktura: Dynamická konektivita na grafu (Holme, De Lichtenberg, Thorup)

- *Jde primárně o zachovávání dvou invariantů*
- *Použijeme-li $a \in \mathcal{O}(\log n), b \in \mathcal{O}(a)$, dostaneme DS s $\mathcal{O}\left(\frac{\log n}{\log \log n}\right)$ worst-case query a $\mathcal{O}\left(\log^2 n\right)$ amort. update*

Zdroje

`The Saga of Minimum Spanning Trees (dizertace Martina Mareše): "5.2 Eulerian Tour trees", "5.3 Dynamic connectivity"`
`https://kam.mff.cuni.cz/~mares/video/zs2021/ga/2021-01-06.mp4`

1.7 Společní předchůdci ve stromech (LCA)

Pozn.: *Lowest Common Ancestor (LCA) úzce souvisí s Range Minimum Query. RMQ lze převést na LCA a to zase lze převést na RMQ$\pm$1 (posloupnost v každém bodě roste nebo klesá o 1). RMQ$\pm$1 pak vyřešíme v $\mathcal{O}(n)$ předzpracování a $\mathcal{O}(1)$ query a tím získáme stejné časy i pro ostatní dva problémy.*

Pozorování: *Pokud se mi nechce předpočítávat, můžu „paralelně“ lézt z obou vrcholů vzhůru – $\mathcal{O}(\min($vzdálenosti těch dvou od LCA$))$*

Algoritmus: Převod LCA na RMQ$\pm$1 *Jednoduchý.*

Algoritmus: Převod RMQ na LCA *Pomocí kartézských stromů.*

Algoritmus: RMQ$\pm$1

- *Dekompozice do bloků (velikosti $b := 1/2 \log n$)*
- *Mikrostruktury uvnitř bloků, makrostruktura nad nimi*
- *Mikrostruktura „bruteforce“ předpočítání všech dvojic – $\mathcal{O}(n^2)$ preprocess, $\mathcal{O}(1)$ query.*
- *Makrostruktura si předpočítá nadbloky obsahující počty bloků odpovídající mocninám dvojky (mohou začínat na kterémkoliv indexu) – $\mathcal{O}(n \log n)$ preprocess, $\mathcal{O}(1)$ query.*
- *Samozřejmě pak analýza časové složitosti.*

Zdroje

`Krajinou grafových algoritmů str 58, 59`

1.8 Minimální kostry

Algoritmus: Kostry z bakaláře *Borůvka, Jarník, Kruskal. Všichni běží v $\mathcal{O}(m \log n)$ a jsou instancemi red-blue algoritmu. Kruskal potřebuje union-find běžící aspoň v $\mathcal{O}(\log n)$.*

Definice:

- *$T[x, y] :=$ cesta v T spojující x, y*
- *Hrana e je T-light iff $\exists f \in T[e] : w(f) > w(e)$*

Věta: *T je minimální kostra iff neexistuje T-light hrana*

Algoritmus: Red-blue metaalgoritmus *Nabarvi nejlehčí hranu elementárního řezu modře nebo nabarvi nejtěžší hranu cyklu červeně (správnost ez snad krom dokázání, že vše bude nabarveno).*

Algoritmus: Kontrahující Borůvka *Zahazuje smyčky. Z paralelních hran vždy zahodí všechny kromě nejlehčí (red rule).*

Je třeba rozmyslet, aby kontrakce v každé fázi proběhly v $\mathcal{O}(m)$. (vypadá ez)

Analýza: Kontrahující Borůvka $\mathcal{O}(\min(n^2, m \log n))$

Analýza: Kontr. Borůvka s omezenou hustotou *Na třídách grafů s konstantou shora omezenou m/n běží v $\mathcal{O}(n)$*

Fakt: *Minor-closed třídy grafů mají omezené m/n. To jsou třeba rovinné grafy.*

Analýza: *Jarník s Fibonacciho haldou $\mathcal{O}(n \log n + m)$, takže pro třídy grafů s $m/n \in \Omega(\log n)$ běží v $\mathcal{O}(m)$*

Myšlenky algoritmu: Fredman-Tarjan *Kombinuje chytře Jarníka a kontrahujícího borůvku $\mathcal{O}(m \cdot \log^* n)$*

Je lin. pro $\exists k : m \in \Omega(n \log^{(k)} n)$.

Stačí jen vědět, že následující alg. existují

Myšlenky algoritmu: Verifikace minimality v $O(m)$ *Používá Borůvka trees, musíme ukázat že stačí lineárně mnoho porovnání vah hran a pak ještě algoritmus, který je v lineárním čase najde*

Myšlenky algoritmu: Karger-Klein-Tarjan *Vychází z verifikace minimality. Expected* $\mathcal{O}(m)$.

Pozn.:

- *Pettie-Ramachandran je optimální ač nikdo neví,* co *je optimální časová složitost. Učili jsme se ho na GA2, ale na státnice bych ho úplně vynechal.*
- *Neučili jsme se:* $\mathcal{O}(m)$ *pro celočíselné váhy, Chazelle* $\mathcal{O}(m \cdot \alpha(m, n))$

1.9 Testování rovinnosti grafů a kreslení do roviny

Pozn.:

- *DFS nám spočítá:*
  - $Enter(v)$ – *kdy DFS vstoupilo*
  - $Ancestor(v)$ – *nejnižší* $Enter(u)$ *pro zpětné hrany vedoucí z* v *do zpět do* u
  - $LowPoint(v)$ – *nejnižší Ancestor podstromu* v
- *2-souvislé „bloky". Mosty v nakreslení zdvojujeme.*
- Externí *je zpětná hrana vedoucí do nenakresleného, její koncový vrchol, jeho blok a všechny bloky a artikulace nad ním.*
- Živý *je koncový vrchol zpětné hrany do zpracovávaného vrcholu jeho blok a všechny bloky a artikulace nad ním.*
- *Externí vrcholy nesmíme překlenout. Tomu se bráníme překlápěním bloků. Děláme to líně, šetříme tak čas, příznaky v artikulacích.*
- *Pro zrychlení: obcházíme jen živý podgraf.*

Algoritmus: Kreslení do roviny v $\mathcal{O}(n)$

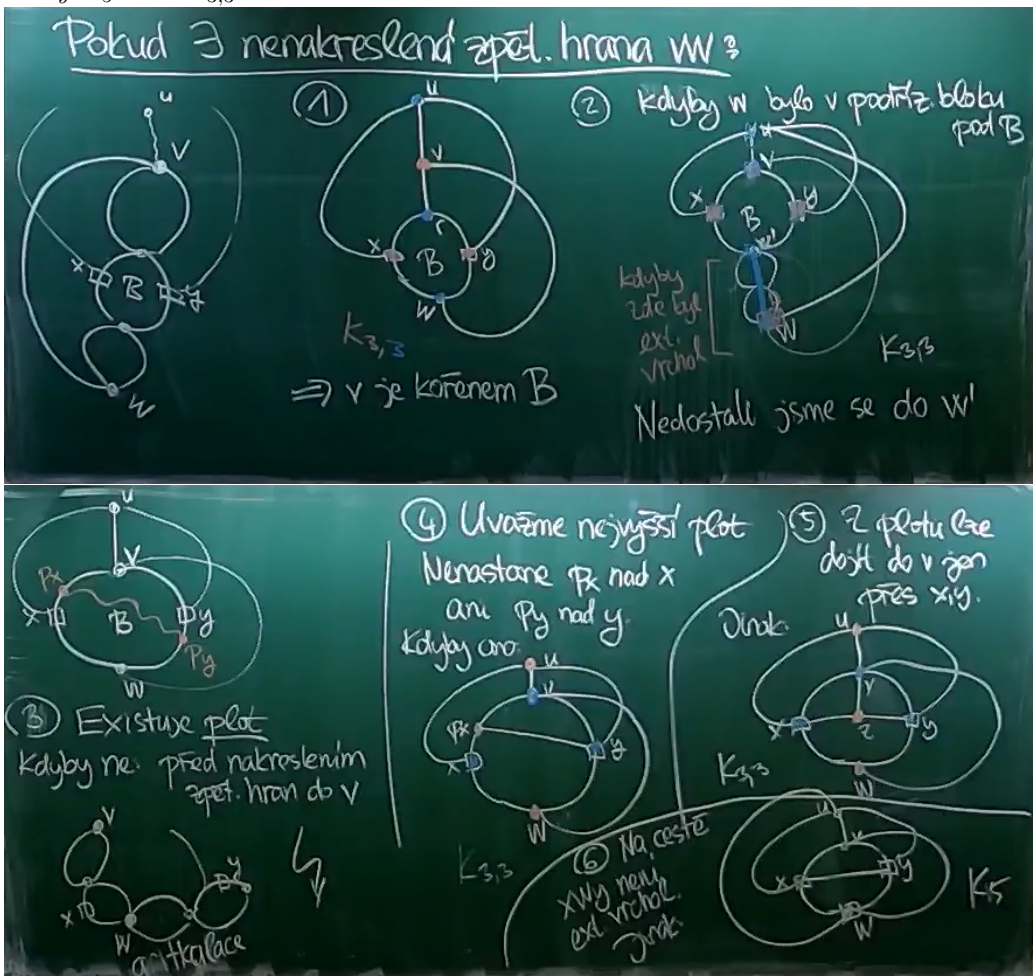
1. *Pokud má graf více než* $3n - 6$ *hran, odmítneme ho rovnou jako nerovinný*
2. *Prohledáme graf* G *do hloubky, spočteme* Enter, Ancestor *a* LowPoint *všech vrcholů.*
3. *Vytvoříme* BlockList *všech vrcholů přihrádkovým tříděním.*
4. *Procházíme vrcholy v pořadí klesajících* Enterů, *pro každý vrchol* v*:*
   (a) *Nakreslíme všechny stromové hrany z* v *jako triviální bloky (2-cykly).*
   (b) *Označíme živý podgraf.*
   (c) *Pro každého syna vrcholu* v *obcházíme živý podgraf náležící k tomuto vrcholu* v *obou směrech a kreslíme zpětné hrany do* v.

(d) Zkontrolujeme, zda všechny zpětné hrany vedoucí do v byly nakresleny, a pokud ne, prohlásíme graf za nerovinný a skončíme.

5. Projdeme hotové nakreslení do hloubky a zorientujeme seznamy sousedů.

Pravidlo 1: V každém živém vrcholu zpracováváme nejdříve zpětné hrany do v, pak podřízené živé interní bloky a konečně podřízené živé externí bloky.
Pravidlo 2: Pokud vstoupíme do dalšího bloku, vybereme si směr, ve kterém budeme pokračovat, následovně: preferujeme směr k živému internímu vrcholu, pokud takový neexistuje, pak k živému externímu vrcholu. Pokud se tento směr liší od směru, ve kterém jsme zatím hranici obcházeli, blok překlopíme.

Myšlenky analýzy: Když alg. nenakreslí zpětnou hranu, graf není rovinný *Ukazujeme, že je přítomný K_5 nebo $K_{3,3}$ minor*



Zdroje

`Krajinou grafových algoritmů: Kapitola 11`

`https://kam.mff.cuni.cz/~mares/video/ls2526/ga2/02-rovinne-kresleni.mp4`

1.10 Union-find

Pozn.: *Používáme n pro počet vrcholů, na kterých pracujeme a m pro počet operací, které provádíme. Pokud začínáme s izolovanými vrcholy, $n \in \mathcal{O}(m)$ a složitosti se pak zjednoduší na $\mathcal{O}(\log n)$ nebo $\mathcal{O}(\alpha(m,n))$ za operaci.*

Analýza: Union by rank $\mathcal{O}(\log n)$ na operaci *Stromeček ranku r má alespoň 2^r vrcholů. (Indukcí, jediný způsob jak zvětšit stromeček ranku $r-1$ na r je unionovat k němu další stromeček ranku $r-1$)*

Analýza: Path compression $\mathcal{O}((n+m)\log n)$

- *Představujeme si, že nejprve provedeme uniony a pak komprese*
- *Indukce přes partition lesu na top a bottom část*
- *Partitionuji na půlky a to mi dá logaritmus*

Definice: Inverzní Ackermanova funkce

- $f^*(n) :=$ *kolikrát je třeba aplikovat f na n, abych dostal* 1
- $\alpha(m,n) :=$ *kolik hvězdiček, abych dostal* $\log^{**\cdots*}(n) \leq \frac{m}{n}$

Pozn.: *Toto je jedna z více možných definic α.*

Myšlenky analýzy: Path compression + union by rank $\mathcal{O}(\alpha(m,n)m+n)$ *Používám stejné lemma jako pro analýzu path compression, ale tentokrát partitionuji nějak podle ranků*

Zdroje

`https://kam.mff.cuni.cz/~mares/video/ls2122/ds2/09-union-find.mp4`

1.11 Párování

Pozn.: *V tomhle podtématu bych se snažil co nejvíc mluvit o tocích. Kromě nich jsem algoritmické teorie párování na magistrovi moc neviděl (vlastně žádnou).*

Pozn.: *Uvažujeme maximální párování co do počtu hran v párování. Maximální co do inkluze lze získat hladově.*

Algoritmus: Maximální párování v bipartitním grafu pomocí toku *Zdůvodnění je látka z bakaláře. Pomocí* $\min(\deg^{in}(v), \deg^{out}(v)) \leq 1$ *Dinice v $\mathcal{O}(\sqrt{n}\cdot m)$. Vážený případ by se taky řešil pomocí toků, ale už ne tak rychle.*

Věta: Kőnigova *V bipartitních grafech velikost největšího párování = velikost nejmenšího pokrytí.*

Myšlenky algoritmu: Blossom algorithm

- *Mělo by snad stačit vědět, jak zhruba funguje a že existuje. Učí se na bakaláři, ne v GA1, GA2 ani DS2.*
- *Mám maximální párování právě tehdy když v grafu není alternující cesta*

- *Při hledání alternujících cest nám nějak překážejí blossoms – liché alternující cykly, z kterých vede zpárovaná hrana*
- *Blossoms kontrahujeme a dekontrahujeme*
- $\mathcal{O}(e \cdot v^2)$

Zdroje

Tomáš Sláma: The Blossom Algorithm
https://www.youtube.com/watch?v=3roPs1Bvg1Q

2 Kdyby toho bylo málo (haha)

2.1 Pokročilé algoritmy pro nejkratší cesty

- Tady je variabilita v tom, jak moc se chci učit důkazy všech těch vět (každý sám o sobě je vlastně v pohodě) a jak detailně se chci učit tu haldu pro délky $\leq L$.

2.2 Tranzitivní uzávěr

- Umět detailně popsat walk algebru a její použití
- Naučit se Seidelův algoritmus do detailu

2.3 Toky v sítích

- S vylepšením, které jsme neprobírali ale je v průvodci, Goldberga dostaneme na $\mathcal{O}(n^2\sqrt{m})$

2.4 Řezy

- Nagamochi-Ibaraki – deterministický $O(mn)$ algoritmus pro nalezení globálního minimálního řezu dokonce na váženém grafu. Je popsaný v Krajinou grafových algoritmů. Ale na GA1, GA2 ani DS2 se v posledních letech neučil.

2.5 Link/cut stromy

- Naučit se amort. analýzu do detailu

2.6 E-T stromy, plně dynamické udržování komponent souvislosti

2.7 Společní předchůdci ve stromech (LCA)

- LCA se dá počítat pomocí HLD. Ale má to horší složitost: $\mathcal{O}(n)$ preprocessing a $\mathcal{O}(\log n)$ query.
- Co dynamické LCA? Prý se na to hodí ET-stromy – píšout to:

  `https://www.ucw.cz/~hubicka/papers/proj/node23.html`
  `https://en.wikipedia.org/wiki/Euler_tour_technique#Applications`

2.8 Minimální kostry

2.9 Testování rovinnosti grafů a kreslení do roviny

- Naučit se analýzu správnosti algoritmu dopodrobna.

2.10 Union-find

- Naučit se analýzy path compression a union-by-rank + path compression detailně.

2.11 Párování

- Naučit se blossom algorithm detailně.