11. Kreslení grafů do roviny

Rovinné grafy se objevují v nejrůznějších praktických aplikacích teorie grafů, a tak okolo nich vyrostlo značné množství algoritmů. I když existují výjimky, jako například již zmíněné hledání kostry rovinného grafu, většina takových algoritmů pracuje s konkrétním vnořením grafu do roviny (rovinným nakreslením).

Proto se zaměříme na algoritmus, který zadaný graf buďto vnoří do roviny, nebo se zastaví s tím, že graf není rovinný. Tarjan již v roce 1974 ukázal [3], že je to možné provést v lineárním čase, ale jeho algoritmus je poněkud komplikovaný. Od té doby se objevilo mnoho zjednodušení, prozatím vrcholících algoritmem Boyera a Myrvoldové [1], a ten zde ukážeme.

Zmiňme ještě, že existují i algoritmy, které vytvářejí rovinná nakreslení s různými speciálními vlastnostmi. Je to například Schnyderův algoritmus [4] generující v lineárním čase nakreslení, v němž jsou hrany úsečkami a vrcholy leží v mřížových bodech mřížky $(n-2)\times(n-2)$, a o něco jednodušší algoritmus Chrobaka a Payneho [2] kreslící do mřížky $(2n-4)\times(n-2)$. Oba ovšem pracují tak, že vyjdou z obyčejného rovinného nakreslení a upravují ho, aby splňovalo dodatečné podmínky.

DFS a bloky

Připomeňme si nejprve některé vlastnosti prohledávání do hloubky (DFS) a jeho použití k hledání komponent vrcholové 2-souvislosti *(bloků)*.

Definice: Prohledávání orientovaného grafu do hloubky rozdělí hrany do čtyř druhů:

- *stromové* – po nich DFS prošlo a rekurzivně se zavolalo; tyto hrany vytvářejí *DFS strom* orientovaný z kořene;
- *zpětné* – vedou do vrcholu, který leží na cestě z kořene DFS stromu do právě prohledávaného vrcholu; jinými slovy vedou do vrcholu, který se právě nachází na zásobníku;
- *dopředné* – vedou do již zpracovaného vrcholu ležícího v DFS stromu pod aktuálním vrcholem;
- *příčné* – zbývající hrany, které vedou do vrcholu již zpracovaného vrcholu v jiném podstromu.

Pozorování: Pokud DFS spustíme na neorientovaný graf a hranu, po níž jsme už jednou prošli, v opačném směru ignorujeme, existují pouze dopředné a zpětné hrany. DFS strom tvoří kostru grafu.

Nyní už se zaměříme pouze na neorientované grafy ...

Lemma: Relace „Hrany e a f leží na společné kružnici“ (značíme $e \sim f$) je ekvivalence. Její třídy tvoří maximální 2-souvislé podgrafy (bloky); ekvivalenční třídy s jedinou hranou (mosty) považujeme za triviální bloky. Vrchol v je artikulace právě tehdy, sousedí-li s ním hrany z více bloků.

Pokud spustíme na graf DFS, je přirozené testovat, do jakých bloků patří stromové hrany sousedící s právě prohledávaným vrcholem v: stromová hrana uv, po které jsme do v přišli, a hrany vw_1 až vw_k vedoucí do podstromů T_1 až T_k (zpětné hrany

jsou vždy ekvivalentní s hranou uv). Pokud je $uv \sim vw_i$, musí existovat cesta z podstromu T_i do vrcholu u, která nepoužije právě testované hrany. Taková cesta ale může podstrom opustit pouze zpětnou hranou (stromová je zakázaná a dopředné ani příčné neexistují). Jinými slovy $uv \sim vw_i$ právě tehdy, když existuje zpětná hrana z podstromu T_i do vrcholu u nebo blíže ke kořeni.

Pokud některá dvojice vw_i, vw_j není ekvivalentní přes hranu uv (nebo pokud hrana uv ani neexistuje, což se nám v kořeni DFS stromu může stát), leží tyto hrany v různých blocích, protože T_i a T_j mohou být spojeny jen přes své kořeny (příčné hrany neexistují). Ze zpětných hran tedy získáme kompletní strukturu bloků.

Nyní si stačí rozmyslet, jak existenci zpětných hran testovat rychle. K tomu se bude hodit:

Definice: Je-li v vrchol grafu, pak:

- *Enter*(v) udává pořadí, v němž DFS do vrcholu v vstoupilo.
- *Ancestor*(v) je nejmenší z *Enter*ů vrcholů, do nichž vede z v zpětná hrana, nebo $+\infty$, pokud z v žádná zpětná hrana nevede.
- *LowPoint*(v) je minimum z *Ancestor*ů vrcholů ležících v podstromu pod v, včetně v samého.

Pozorování: *Enter*, *Ancestor* i *LowPoint* všech vrcholů lze spočítat během prohledávání, tedy také v lineárním čase.

Rozpoznávání bloků a artikulací můžeme shrnout do následujícího lemmatu:

Lemma: Pokud v je vrchol grafu, u jeho otec a w jeho syn v DFS stromu, pak stromové hrany uv a vw leží v tomtéž bloku ($uv \sim vw$) právě tehdy, když *LowPoint*$(w) <$ *Enter*(v), a v je artikulace právě když některý z jeho synů w má *LowPoint*$(w) \geq$ *Enter*(v). Kořen DFS stromu je artikulace, právě když má více než jednoho syna.

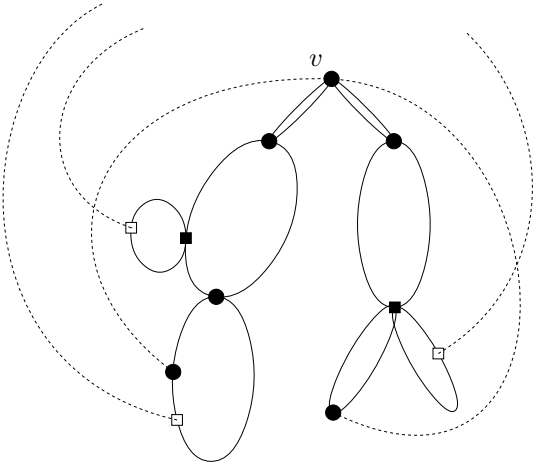
Postup kreslení

Graf budeme kreslit v opačném pořadí oproti DFS, tj. od největších *Enter*ů k nejmenším, a vždy si budeme udržovat blokovou strukturu již nakreslené části grafu, uspořádanou podle DFS stromu – každý blok bude mít svůj kořen, s výjimkou nejvyššího bloku je tento kořen současně artikulací v nadřazeném bloku. Aby se nám tato situace snadno reprezentovala, artikulace naklonujeme a každý blok dostane svou vlastní kopii artikulace.

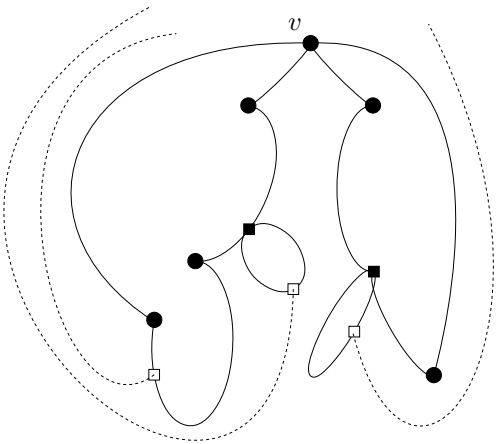
Také budeme využívat toho, že nakreslení každého bloku, který není most, je ohraničeno kružnicí, a mosty zdvojíme, aby to pro ně platilo také. Navíc v libovolném nakreslení můžeme kterýkoliv blok spolu se všemi bloky ležícími pod ním překlopit podle kořenové artikulace, aniž bychom porušili rovinnost.

Všimněme si, že pokud vede z nějakého už nakresleného vrcholu ještě nenakreslená hrana, lze pokračovat po nenakreslených hranách až do kořene DFS stromu. Všechny vrcholy, ke kterým ještě bude potřeba něco připojit (takovým budeme říkat *externí* a hranám rovněž; za chvíli to nadefinujeme formálně), proto musí ležet v téže stěně dosud nakresleného podgrafu a bez újmy na obecnosti si vybereme, že to bude vnější stěna.

Základním krokem algoritmu tedy bude rozšířit nakreslení o nový vrchol v a o všechny hrany vedoucí z něj do jeho (již nakreslených) DFS-následníků. Stromové hrany půjdou nakreslit vždy, přidáme je jako triviální bloky (2-cykly) a nejsou-li to mosty, brzy se nějakou zpětnou hranou spojí s jinými bloky. Zpětné hrany byly až donedávna externí, takže přidání jedné zpětné hrany nahradí cestu po okraji bloku touto hranou (tím vytvoří novou stěnu) a také může sloučit několik bloků do jednoho, jak je vidět z obrázků.



Před nakreslením zpětných hran ...



... po něm (čtverečky jsou externí vrcholy)

Bude se nám hodit, že čas potřebný na tuto operaci je přímo úměrný počtu hran, které ubyly z vnější stěny, což je amortizovaně konstanta.

Může se nám ale také stát, že zpětná hrana zakryje nějaký externí vrchol. Tehdy musíme některé bloky překlopit tak, aby externí vrcholy zůstaly venku. Potřebujeme tedy datové struktury, pomocí nichž bude možné překlápět efektivně a co víc, také rychle poznávat, kdy je překlápění potřebné.

Externí vrcholy

Jestliže z nějakého vrcholu v bloku B vede dosud nenakreslená hrana, musí být tento vrchol na vnější stěně, takže musí také zůstat na vnější stěně i vrchol, přes který je B připojen ke zbytku grafu. Proto externost nadefinujeme tak, aby pokrývala i tyto případy:

Definice: Vrchol w je *externí,* pokud buďto z w vede zpětná hrana do ještě nenakresleného vrcholu, nebo je pod w připojen externí blok, čili blok obsahující alespoň jeden externí vrchol. Ostatním vrcholům budeme říkat *interní.*

Jinými slovy platí, že vrchol w je externí při zpracování vrcholu v, pakliže $\mathit{Ancestor}(w) < \mathit{Enter}(v)$, nebo pokud pro některého ze synů x ležícího v jiném bloku je $\mathit{LowPoint}(x) < \mathit{Enter}(v)$. Druhá podmínka funguje díky tomu, že kořen bloku má v tomto bloku právě jednoho syna (jinak by existovala příčná hrana, což víme, že není pravda), takže minimum z *Ancestor*ů všech vrcholů ležících uvnitř bloku je přesně *LowPoint* tohoto syna. Ve statickém grafu by se všechny testy redukovaly na $\mathit{LowPoint}(w)$, nám se ovšem bloková struktura průběžně mění, takže musíme uvažovat bloky v současném okamžiku. Proto si zavedeme:

Definice: $\mathit{BlockList}(w)$ je seznam všech bloků připojených v daném okamžiku pod vrcholem w, reprezentovaných jejich kořeny (klony vrcholu w) a jedinými syny kořenů. Tento seznam udržujeme setříděný vzestupně podle *LowPoint*ů synů.

Lemma: Vrchol w je externí při zpracování vrcholu v, pokud je buďto $\mathit{Ancestor}(w) < \mathit{Enter}(v)$, nebo první prvek seznamu $\mathit{BlockList}(w)$ má $\mathit{LowPoint} < \mathit{Enter}(v)$. Navíc seznamy *BlockList* lze udržovat v amortizovaně konstantním čase.

Důkaz: První část plyne přímo z definic. Všechny seznamy na začátku běhu algoritmu sestrojíme v lineárním čase přihrádkovým tříděním a kdykoliv sloučíme blok s nadřazeným blokem, odstraníme ho ze seznamu v příslušné artikulaci. ♡

Reprezentace bloků a překlápění

Pro každý blok si potřebujeme pamatovat vrcholy, které leží na hranici (některé z nich jsou externí, ale to už umíme poznat) a bloky, které jsou pod nimi připojené. Dále ještě vnitřní strukturu bloku včetně uvnitř připojených dalších bloků, ale jelikož žádné vnitřní vrcholy nejsou externí, vnitřek už neovlivní další výpočet a potřebujeme jej pouze pro vypsání výstupu.

Pro naše účely bude stačit zapamatovat si u každého bloku, jestli je oproti nadřazenému bloku překlopen. Tuto informaci zapíšeme do kořene bloku. Každý vrchol na hranici bloku pak bude obsahovat dva ukazatele na sousední vrcholy. Neumíme sice lokálně poznat, který ukazatel odpovídá kterému směru, ale když se nějakým směrem vydáme, dokážeme ho dodržet – stačí si vždy vybrat ten ukazatel, který nás nezavede do právě opuštěného vrcholu.

Každý vrchol si také bude pamatovat seznam svých sousedů, podle orientace bloku buďto v hodinkovém nebo opačném pořadí. Chceme-li přidat hranu, potřebujeme tedy znát absolutní orientaci, ale to půjde snadno, jelikož hrany přidáváme jen k vrcholům na hranici, poté co k nim po hranici dojdeme z kořene.

K překlopení bloku včetně všech podřízených bloků nám stačí invertovat bit v kořeni, pokud chceme překlopit jen tento blok, invertujeme bity i v kořenech všech podřízených bloků, jež najdeme obcházením hranice.

Na konci algoritmu spustíme dodatečný průchod, který všechny překlápěcí bity přenese ve směru od kořene k potomkům a určí tak absolutní orientaci všech seznamů sousedů i hranic.

Živý podgraf

Když nakreslíme nový vrchol v a z něj vedoucí stromové hrany, musíme obejít každý podstrom, ve vhodném pořadí nakreslit zpětné hrany do v a podle potřeby překlopit bloky. V podstromu ovšem může být mnoho bloků, které žádnou pozornost nevyžadují a běh algoritmu by zbytečně brzdily. Proto si před samotným kreslením označíme *živou* část grafu – to je část, kterou potřebujeme během kreslení navštívit; zbytku grafu se budeme snažit vyhýbat, aby nás nezdržoval.

Definice: Vrchol w je *živý,* pokud z něj buďto vede zpětná hrana do právě zpracovávaného vrcholu v, nebo pokud pod ním je připojen živý blok, tj. blok obsahující živý vrchol.

Živé vrcholy přitom mohou být i externí (pokud z nich vedou zpětné hrany jak do vrcholu v, tak do ještě nenakreslených vrcholů). Pokud nějaký vrchol není ani živý, ani externí, budeme ho nazývat *pasivní.*

Před procházením podstromů tedy nejprve probereme všechny zpětné hrany vedoucí do v a označíme živé vrcholy. Pro každou zpětnou hranu potřebujeme oživit vrchol, z nějž hrana vede, dále artikulaci, pod níž je tento blok připojen, a další artikulace na cestě do v. Tedy pokaždé, když vstoupíme do bloku (nějakým vrcholem na vnější stěně), potřebujeme nalézt kořen bloku. To uděláme tak, že začneme obcházet vnější stěnu oběma směry současně, až dojdeme v některém směru do kořene. Navíc si všechny vrcholy, přes něž jsme prošli, označkujeme a přiřadíme k nim rovnou ukazatel na kořen, tudíž po žádné části hranice neprojdeme vícekrát.[1]

Výstupem této části algoritmu budou značky u živých vrcholů a u artikulací také seznamy podřízených živých bloků. Tyto seznamy budeme udržovat uspořádané tak, aby externí bloky následovaly po všech interních. To nám usnadní práci v hlavní části algoritmu.

Lemma: Pro každý kořen trvá značení živých vrcholů čas $\mathcal{O}(k + \ell)$, kde k je počet kreslených zpětných hran a ℓ počet hran, které zmizely z vnější stěny, čili amortizovaně konstanta.

[1] Značky ani nebude potřeba mazat, když si u nich poznamenáme, který vrchol byl kořenem v okamžiku, kdy jsme značku vytvořili, a značky patřící ke starým kořenům budeme ignorovat, resp. přepisovat.

Důkaz: Po žádné hraně neprojdeme více než jednou. Navíc alespoň polovina z těch, po nichž jsme prošli, zmizí z vnější stěny, takže hledání kořenů bloků trvá $\mathcal{O}(\ell)$. Pro každou zpětnou hranu označíme jeden vrchol jako živý a pak pokračujeme hledáním kořenů, které jsme již započítali. ♡

Kreslení zpětných hran

Nyní již máme vše připraveno – datové struktury, detekci externích vrcholů a označování živého podgrafu – a zbývá doplnit, jak algoritmus kreslí zpětné hrany. Jelikož zpětné hrany vedoucí do v nemohou způsobit sloučení bloků ležících pod v (na to jsou potřeba zpětné hrany vedoucí někam nad v a ty ještě nekreslíme), zpracováváme každý podstrom zvlášť. Vždy přidáme triviální blok pro stromovou hranu, pod něj připojíme blokovou strukturu zatím nakreslené části podstromu a vydáme se po hranici této struktury nejdříve jedním a pak druhým směrem.

Oba průchody vypadají následovně: Procházíme seznam vrcholů na hranici a pasivní vrcholy přeskakujeme. Pokud objevíme živý vrchol, nakreslíme vše, co z něj vede, případně se zanoříme do živých bloků, které jsou připojeny pod tímto vrcholem. Pokud objevíme externí vrchol (poté, co jsme ho případně ošetřili jako živý), procházení zastavíme, protože za externí vrchol již nemůžeme po této straně hranice nic připojit, aniž by se externí vrchol dostal dovnitř nakreslení.

Přitom se řídíme dvěma jednoduchými pravidly:

Pravidlo #1: V každém živém vrcholu zpracováváme nejdříve zpětné hrany do v, pak podřízené živé interní bloky a konečně podřízené živé externí bloky. (K tomu se nám hodí, že máme seznamy živých podřízených bloků setříděné.)

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.

Časová složitost této části algoritmu je lineární ve velikosti živého podgrafu až na dvě výjimky. Jednou je konec prohledávání od posledního živého vrcholu k bodu zastavení, druhou pak vybírání strany hranice při vstupu do bloku. V obou můžeme procházet až lineárně mnoho pasivních vrcholů. Pomůžeme si ovšem snadno: kdykoliv projdeme souvislý úsek hranice tvořený pasivními vrcholy, přidáme pomocnou hranu, která tento úsek překlene. Můžeme ji dokonce přidat do nakreslení a podrozdělit si tak vnější stěnu.

Hotový algoritmus

Algoritmus (kreslení do roviny):

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 *Enter*ů, pro každý vrchol v:

5. Nakreslíme všechny stromové hrany z v jako triviální bloky (2-cykly).
6. Označíme živý podgraf.
7. Pro každého syna vrcholu v obcházíme živý podgraf náležící k tomuto vrcholu v obou směrech a kreslíme zpětné hrany do v.
8. Zkontrolujeme, zda všechny zpětné hrany vedoucí do v byly nakresleny, a pokud ne, prohlásíme graf za nerovinný a skončíme.
9. Projdeme hotové nakreslení do hloubky a zorientujeme seznamy sousedů.

Věta: Tento algoritmus pro každý graf doběhne v čase $\mathcal{O}(n)$ a pokud byl graf rovinný, vydá jeho nakreslení, v opačném případě ohlásí nerovinnost.

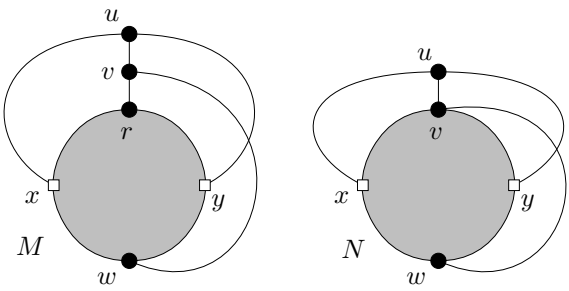
Důkaz: První krok je korektní, jelikož pro všechny rovinné grafy je $m \leq 3n-6$; nadále tedy můžeme předpokládat, že $m = \mathcal{O}(n)$. Lineární časovou složitost kroků 4–6 a 9 jsme již diskutovali, kroky 7–8 jsou lineární ve velikosti živého podgrafu, a tedy také $\mathcal{O}(n)$. Nakreslení vydané algoritmem je vždy rovinné a všechny stromové hrany jsou vždy nakresleny, zbývá tedy ukázat, že zpětnou hranu můžeme nenakreslit, jen pokud graf nebyl rovinný. Tomu věnujeme zbytek kapitoly. ♡

Důkaz korektnosti

Lemma: Pokud existuje zpětná hrana, kterou algoritmus nenakreslil, graf na vstupu není rovinný.

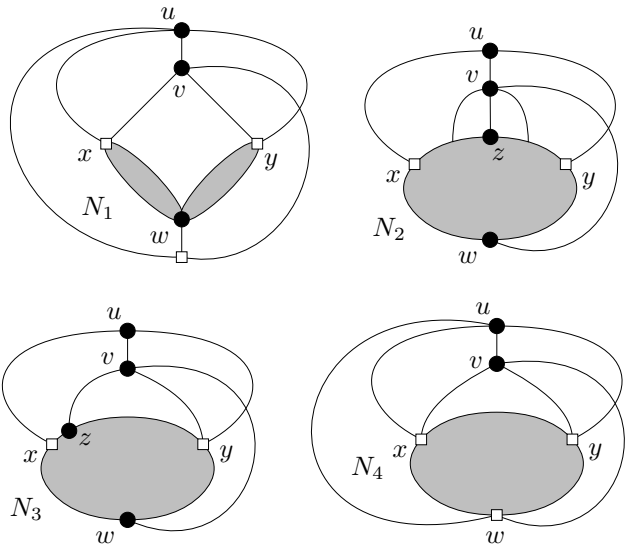
Důkaz: Pro spor předpokládejme, že po zpracování vrcholu v existuje nějaká zpětná hrana wv, kterou algoritmus nenakreslil, čili že přístup z v k w je v obou směrech blokován externími vrcholy. Rozborem případů ukážeme, že tato situace vede ke sporu buďto s pravidly #1 a #2 nebo s rovinností grafu.

Označme B blok, ve kterém leží na obou stranách hranice nějaké externí vrcholy x a y a pod nimi je připojen nějakou cestou vrchol w. Takový blok musí určitě existovat, protože jinak by algoritmus všechny bloky na cestě z v do w popřeklápěl tak, aby se hrana wv vešla. V grafu se tedy musí vyskytovat jeden z následujících minorů (do vrcholu u jsme kontrahovali celou dosud nenakreslenou část grafu; vybarvená část odpovídá vnitřku bloku; hranaté vrcholy jsou externí):



Minor M přitom odpovídá situaci, kdy v neleží v bloku B. Tento případ snadno

vyloučíme, protože M je isomorfní s grafem $K_{3,3}$. V grafu se proto musí vyskytovat N. Tento minor je ale rovinný, takže musíme ukázat, že vnitřek bloku brání nakreslení hrany vw dovnitř. Vždy pak dojdeme k některému z následujících nerovinných minorů (N_1 až N_3 jsou isomorfní s $K_{3,3}$ a N_4 s K_5):



Uvažme, jak bude B vypadat po odebrání vrcholu v a hran z něj vedoucích:

a) *přestane být 2-souvislý* – tehdy se zaměříme na bloky ležící na cestě xy:

   1) w *je artikulace* na této cestě – BÚNO je taková artikulace v DFS prohledána po bloku obsahujícím x, ale před y. Tehdy nám jistě x nezabránilo v tom, abychom do w došli (může blokovat jenom jednu stranu hranice), takže jsme se ve w museli rozhodnout, že přednostně zpracujeme pokračování cesty do y před hranou vw, a to je spor s pravidlem #1.
   2) w *je v bloku připojeném pod takovou artikulací* – aby se pravidlo #1 vydalo do y místo podřízených bloků, musí být alespoň jeden z nich externí, takže v G je minor N_1.
   3) w *je v bloku na cestě nebo připojen pod takový blok* – opět si všimneme, že do bloku jsme vstoupili mezi x a y. Abychom se podle pravidla #2 rozhodli pro stranu, z níž nevede hrana vw, musela na druhé straně být také hrana do v, a proto se v grafu vyskytuje minor M.

b) *zůstane 2-souvislý* a vznikne z něj nějaký blok B' – tehdy rozebereme, jaké hrany vedou mezi v a B':

   1) *více než dvě hrany* – minor N_2.

2) *alespoň jedna hrana na „horní“ cestu* (to jest na tu, na niž neleží w) – minor N_3.
3) *dvě hrany do x, y nebo na „dolní“ cestu* – ať už jsme vstoupili na hranici bloku B' kteroukoliv hranou, pravidlo #2 nám řeklo, že máme pokračovat vrchem, což je možné jedině tehdy, je-li na spodní cestě ještě jeden externí vrchol, a to dává minor N_4. ♡

Poznámka: Podle tohoto důkazu bychom také mohli v lineárním čase v každém nerovinném grafu nalézt Kuratowského podgraf, dokonce také v $O(n)$, jelikož když je $m > 3n - 6$, můžeme se omezit na libovolných $3n - 5$ hran, které určitě tvoří nerovinný podgraf.

Literatura

[1] J. Boyer and W. Myrvold. On the cutting edge: Simplified $\mathcal{O}(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[2] M. Chrobak and T. H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Information Processing Letters*, 54(4):241–246, 1995.

[3] J. Hopcroft and R. Tarjan. Efficient Planarity Testing. *Journal of the ACM (JACM)*, 21(4):549–568, 1974.

[4] W. Schnyder. Embedding planar graphs on the grid. *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 138–148, 1990.