Kapitola 4

Reprezentace grafů a základní algoritmy

Velké množství problémů v informatice se formuluje pomocí grafových pojmů. Abychom tyto problémy mohli řešit na počítači, musíme mít k dispozici takovou reprezentaci grafu, která je vhodná pro počítačové zpracování. Mezi vhodné reprezentace z tohoto hlediska nemůžeme počítat právě vizuální znázornění grafu pomocí obrázku, které jsme pro ilustraci pojmů a vlastností grafů dosud používali. Struktura grafu je totiž zcela vyjádřena v jeho incidenci, tedy v určení krajních uzlů hran. Naproti tomu např. bitová mapa zachycující obrázek grafu je vedle dalších nevýhod i zbytečně paměťově náročná a pro reprezentaci jeho struktury nevhodná. Obsahuje totiž převážně neužitečná data o konkrétním tvaru a rozmístění grafických prvků (kroužky a oblouky nebo úsečky) vyjadřujících uzly a hrany grafu, která mohou mít pro jiný obrázek téhož grafu zcela odlišné hodnoty.

To ovšem neznamená, že bychom při řešení grafových úloh nikdy neuvažovali vedle strukturní i případnou vizuální podobu grafu. Tyto případy jsou však omezeny jen na speciální grafovou problematiku (např. planární grafy) nebo aplikace (např. propojování prvků na masce integrovaného obvodu nebo na desce tištěných spojů). Vizuální znázornění struktury grafu je ovšem vhodné také jako prostředek styku s lidským uživatelem, můžeme je tedy považovat nejvýše za formu vnější a nikoliv vnitřní reprezentace.

4.1 Reprezentace grafů

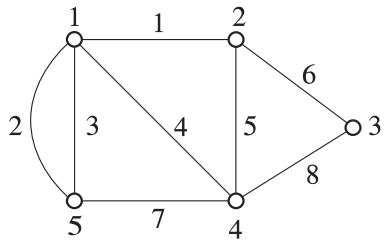
Existují dva druhy reprezentace grafu: maticová a spojová. Maticový popis grafů má spíše teoretický nežli praktický význam. Poskytuje totiž vazbu mezi teorií grafů a lineární algebrou, jež dává vedle možnosti charakterizovat vlastnosti grafů algebraickými prostředky i řadu zajímavých tvrzení (např. určení počtu koster úplného grafu pomocí Cauchyova-Binetova vzorce).

Různé varianty spojové reprezentace jsou standardním způsobem vyjádření grafu při realizaci grafových algoritmů. Oproti maticové formě mají jednak menší asymptotickou paměťovou složitost, ale především umožňují dosáhnout pro většinu řešených úloh i lepší časové složitosti algoritmů ve srovnání s maticovou reprezentací.

Matice neorientovaných grafů

Definice 4.1: Nechť $G = \langle H, U, \varrho \rangle$ je neorientovaný graf s množinou hran $H = \{h_1, h_2, \ldots, h_m\}$ a množinou uzlů $U = \{u_1, u_2, \ldots, u_n\}$. **Maticí incidence** grafu G nazýváme matici $\mathbf{A} = [a_{ik}]$ typu (n, m) nad tělesem $\mathbf{Z}_2$ (mod 2), jejíž prvky jsou dány vztahem

$$a_{ik} = \begin{cases} 1 & \text{pokud hrana } h_k \text{ inciduje s uzlem } u_i, \\ 0 & \text{v opačném případě.} \end{cases}$$



$A =$

	1	2	3	4	5	6	7	8
1	1	1	1	1	0	0	0	0
2	1	0	0	0	1	1	0	0
3	0	0	0	0	0	1	0	1
4	0	0	0	1	1	0	1	1
5	0	1	1	0	0	0	1	0

$V =$

	1	2	3	4	5
1	0	1	0	1	2
2	1	0	1	1	0
3	0	1	0	1	0
4	1	1	1	0	1
5	2	0	0	1	0

Obrázek 4.1: Matice incidence a sousednosti neorientovaného grafu

S ohledem na studium vlastností incidenční matice je podstatné, že se uvažuje jako matice hodnot nad tělesem $\mathbf{Z}_2$, a nikoliv tedy jako celočíselná nebo booleovská. Nad tělesem $\mathbf{Z}_2$ můžeme totiž zavést vektorový prostor a používat v běžném významu známé pojmy lineární algebry – např. lineární kombinace vektorů, lineárně závislé a nezávislé vektory, hodnost matice, determinant matice apod.[1] Pro jednoduchost připomeneme pravidla počítání v tělese $\mathbf{Z}_2$:

$$\begin{array}{cccc} 0+0=0 & 0+1=1 & 1+0=1 & 1+1=0 \\ 0\cdot 0=0 & 0\cdot 1=0 & 1\cdot 0=0 & 1\cdot 1=1 \end{array}$$

Vytvoření matice incidence ilustrujeme na grafu z obr. 4.1, v němž jsme pro jednoduchost jak uzly, tak hrany označili přímo odpovídajícími čísly. Matice $\mathbf{A}$ (v obrázku zapsaná schematicky formou tabulky) má tolik řádků, kolik má graf G uzlů (5), počet sloupců je roven počtu hran grafu G (8). Řádek odpovídající i-tému uzlu u_i obsahuje právě $\delta_G(u_i)$ jedniček, a ty jsou umístěny ve sloupcích, jež odpovídají hranám incidujícím s uzlem u_i. Každý sloupec matice $\mathbf{A}$ obsahuje právě dvě jedničky, neboť každá hrana inciduje se dvěma uzly (smyčky neuvažujeme). Tvar matice incidence záleží pochopitelně na pořadí, v jakém jsme očíslovali uzly a hrany grafu. Změna očíslování však způsobí pouze permutaci řádků nebo sloupců matice $\mathbf{A}$, takže dostáváme následující tvrzení.

Věta 4.2: Dva neorientované grafy jsou izomorfní právě tehdy, jsou-li jejich matice incidence stejné až na případnou permutaci řádků a sloupců.

Jiná vlastnost grafu, kterou může matice incidence na první pohled odrážet, je rozčlenění grafu na komponenty. Postupujeme-li totiž při očíslování uzlů a hran systematicky podle jednotlivých komponent, pak bude mít $\mathbf{A}$ tvar matice rozdělené na $p \times p$ polí (p je počet komponent)

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{A}_{22} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{A}_{pp} \end{bmatrix}, \tag{4.1}$$

přičemž diagonální pole $\mathbf{A}_{ii}$ pro $i = 1, 2, \dots, p$ představují matice incidence jednotlivých komponent a ostatní pole jsou nulové matice. Pokud ovšem není zajištěno žádoucí očíslování uzlů a hran při vytváření incidenční matice, nedostaneme pro nesouvislý graf přímo matici $\mathbf{A}$ ve tvaru (4.1). Zjišťovat počet a složení komponent zkoumáním všech možných permutací řádků a sloupců matice $\mathbf{A}$ je však pro praktické účely nevhodné, mnohem efektivnější algoritmus pro tuto úlohu naznačíme v dalším textu. V následujících tvrzeních si všimneme blíže algebraických vlastností incidenčních matic.

Věta 4.3: Hodnost matice incidence $\mathbf{A}$ neorientovaného grafu $G = \langle H, U, \varrho \rangle$ je menší než počet jeho uzlů $|U|$.

[1] S vektorovým prostorem nad tělesem reálných nebo komplexních čísel je čtenář jistě v hrubých rysech seznámen a přechod na těleso $\mathbf{Z}_2$ nevyžaduje speciální přípravu. V případě potřeby odkazujeme na [3], [4] nebo [34]. Je vhodné si uvědomit, že odčítat zde znamená totéž, co přičítat, také dělení a násobení jedničkou je stejné.

Důkaz: Hodnost matice je definována jako maximální počet lineárně nezávislých řádků matice. Základní lineární kombinace všech řádkových vektorů matice $\mathbf{A}$ uvažovaných s koeficientem 1 dává nulový vektor (v každém sloupci matice jsou právě dvě jedničky a sčítáme mod 2). Všechny jeho řádky nemohou tedy být lineárně nezávislé. △

Věta 4.4: Nechť $G = \langle H, U, \varrho \rangle$ je souvislý neorientovaný graf. Potom libovolných r $(0 < r < |U|)$ řádků matice incidence $\mathbf{A}$ grafu G je lineárně nezávislých.

Důkaz: Postupujeme sporem. Nechť existuje r řádkových vektorů $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_r$ lineárně závislých, neboli existují koeficienty $\lambda_1, \lambda_2, \ldots, \lambda_r$ ne všechny rovné nule tak, že platí

$$\lambda_1 \mathbf{a}_1 + \lambda_2 \mathbf{a}_2 + \ldots + \lambda_r \mathbf{a}_r = 0. \tag{4.2}$$

Jsou-li některé z λ_i rovny nule, můžeme příslušné vektory ze soustavy vypustit a zbývající vektory budou stále lineárně závislé. Předpokládejme proto, že ve výrazu (4.2) jsou všechna $\lambda_i = 1$ pro $i = 1, 2, \ldots, r$. Je tedy součtem vektorů $\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_r$ nulový vektor (bez újmy na obecnosti můžeme předpokládat, že je to prvních r řádků matice $\mathbf{A}$). V matici $\mathbf{A}$ nyní provedeme permutaci sloupců tak, aby v levé části byly sloupce obsahující ve svých prvních r řádcích dvě jedničky, a v pravé části byly sloupce obsahující své dvě jedničky ve spodních $|U| - r$ řádcích:

$$\mathbf{A} = \begin{bmatrix} \boxed{\mathbf{A}_{11}} & \begin{matrix} \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \ldots & \mathbf{0} \end{matrix} \\ \begin{matrix} \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \ldots & \mathbf{0} \end{matrix} & \boxed{\mathbf{A}_{22}} \end{bmatrix}. \tag{4.3}$$

Mohou zřejmě nastat dva případy:
(a) Jedna z částí $\mathbf{A}_{11}$ nebo $\mathbf{A}_{22}$ se vůbec v matici $\mathbf{A}$ nevyskytuje, pak by ale graf G obsahoval izolované uzly, což odporuje jeho souvislosti.

(b) Matice $\mathbf{A}$ obsahuje obě části $\mathbf{A}_{11}$ a $\mathbf{A}_{22}$, pak ovšem žádný z prvních r uzlů není spojen hranou se žádným ze zbývajících $|U| - r$ (> 0) uzlů, a to opět odporuje předpokladu souvislosti grafu G. △

Důsledek 1: Hodnost matice incidence $\mathbf{A}$ souvislého neorientovaného grafu $G = \langle H, U, \varrho \rangle$ je rovna $|U| - 1$.

Důsledek 2: Pro hodnost matice incidence $\mathbf{A}$ neorientovaného grafu $G = \langle H, U, \varrho \rangle$ o p komponentách platí vztah

$$\mathrm{h}(\mathbf{A}) = |U| - p. \tag{4.4}$$

Tento důsledek poskytuje návod, jak zjistit algebraicky počet komponent grafu ze zadané matice incidence: Nějakým vhodným postupem (např. Gaussovou eliminací) zjistíme hodnost $\mathrm{h}(\mathbf{A})$ matice $\mathbf{A}$, a počet komponent grafu je pak roven rozdílu $p = |U| - \mathrm{h}(\mathbf{A})$. Efektivnější způsob určení počtu (a současně i složení) komponent grafu představuje použití algoritmu systematického průchodu všemi uzly a hranami grafu, který uvedeme v další kapitole.

Definice 4.5: Nechť $G = \langle H, U, \varrho \rangle$ je neorientovaný graf s množinou uzlů $U = \{u_1, u_2, \ldots, u_n\}$. **Maticí sousednosti** grafu G nazýváme čtvercovou matici $\mathbf{V} = [v_{ik}]$ řádu n, jejíž prvky jsou celá čísla daná vztahem

$$v_{ik} = |\varrho^{-1}([u_i, u_k])|. \tag{4.5}$$

Prvek v_{ik} tedy určuje počet hran incidujících s neuspořádanou dvojicí uzlů $[u_i, u_k]$. Matice sousednosti prostého grafu obsahuje pouze prvky 0 a 1, takže se shoduje s maticovou reprezentací relace sousednosti Γ. Díky symetrii této relace bude i matice $\mathbf{V}$ symetrická, tzn. $\mathbf{V} = \mathbf{V}^T$. Na rozdíl od matice incidence nezachycuje matice sousednosti vzájemný vztah hran a uzlů grafu, dovoluje však vyjádřit případnou existenci smyček (nenulovými diagonálními prvky). Příklad matice sousednosti ukazuje pro případ multigrafu obr. 4.1.

Některé vlastnosti matic sousednosti jsou stejné jako u matic incidence – to platí např. pro tvrzení o izomorfismu (s tím rozdílem, že se musí uvažovat pouze stejné permutace množiny řádků i sloupců, takže celkový počet možností je pouze $n!$). Vhodně zvoleným očíslováním množiny uzlů nesouvislého grafu o p komponentách lze rovněž dosáhnout rozkladu matice $\mathbf{V}$ do tvaru matice rozdělené na $p \times p$ polí,

$$\mathbf{V} = \begin{bmatrix} \mathbf{V}_{11} & \mathbf{O} & \dots & \mathbf{O} \\ \mathbf{O} & \mathbf{V}_{22} & \dots & \mathbf{O} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{O} & \mathbf{O} & \dots & \mathbf{V}_{pp} \end{bmatrix}, \tag{4.6}$$

kde diagonální pole $\mathbf{V}_{ii}, i = 1, 2, \dots, p$ jsou matice sousednosti jednotlivých komponent. Hodnost samotné matice $\mathbf{V}$ nemá vztah k souvislosti grafu, platí však následující tvrzení, které dovoluje zjistit počet i složení všech komponent grafu pomocí jeho matice sousednosti $\mathbf{V}$.

Věta 4.6: Nechť $\mathbf{V}$ je matice sousednosti neorientovaného grafu $G = \langle H, U, \varrho \rangle$. Potom prvek $v_{ik}^{(r)}$ r-té mocniny $\mathbf{V}^r$ matice $\mathbf{V}$ udává počet sledů délky r mezi uzly u_i a u_k.

Důkaz: (indukcí podle r)
1. Pro $r = 1$ tvrzení zjevně platí, neboť sled délky 1 je tvořen jedinou hranou.
2. Nechť tvrzení platí pro nějaké $r \geq 1$. Potom je $\mathbf{V}^{r+1} = \mathbf{V}^r \cdot \mathbf{V} = [v_{ik}^{(r+1)}]$, kde

$$v_{ik}^{(r+1)} = \sum_{j=1}^{|U|} v_{ij}^{(r)} v_{jk}. \tag{4.7}$$

Prvek $v_{ij}^{(r)}$ určuje (podle indukčního předpokladu) počet sledů délky r mezi uzly u_i, u_j a prvek v_{jk} počet hran mezi uzly u_j, u_k, takže součin $v_{ij}^{(r)} v_{jk}$ určuje počet možných sledů délky $r+1$ z u_i do u_k vedoucích přes uzel u_j. Sečtením přes všechny možné uzly u_j ve vztahu (4.7) dostáváme tedy celkový počet sledů délky $r + 1$ z uzlu u_i do uzlu u_k. △

Uvažujme nyní reflexivně-tranzitivní uzávěr Γ^* relace sousednosti Γ grafu G. Díky symetrii relace Γ je uzávěr Γ^* ekvivalencí na množině uzlů – je to právě relace dosažitelnosti, která podle cvičení 2.1-16 indukuje rozklad grafu na komponenty. Maticovou reprezentaci $\mathbf{R} = [r_{ik}]$ této relace, která je ekvivalencí na množině uzlů grafu G, nazýváme **maticí dosažitelnosti** neorientovaného grafu G a její prvky mají následující význam:

$$r_{ik} = \begin{cases} 1 & \text{pokud existuje sled mezi uzly } u_i, u_k, \\ 0 & \text{v opačném případě.} \end{cases}$$

K určení matice dosažitelnosti z matice sousednosti grafu můžeme použít matici

$$\mathbf{V}^* = \sum_{i=0}^{d} \mathbf{V}^i, \quad \text{kde } d = \min(|H|, |U| - 1), \tag{4.8}$$

jejíž prvky v_{ij}^* udávají podle věty 4.6 počet sledů délky nejvýše d mezi uzly u_i, u_j. Jednodušší než sčítat mocniny matice $\mathbf{V}$ je ale využít vztahu $R = \Gamma^*$ a počítat matici $\mathbf{R}$ jako matici reflexivně-tranzitivního uzávěru relace Γ pomocí Floydova-Warshallova algoritmu z odst. 5.4.

Matice incidence $\mathbf{A}$ i matice sousednosti $\mathbf{V}$ poskytují o daném grafu téměř stejnou informaci, naproti tomu v matici dosažitelnosti $\mathbf{R}$ je obsažena pouze informace globálního charakteru, z níž se lokální vlastnosti grafu určit nedají. Je možné zabývat se otázkou vzájemného vztahu matic $\mathbf{A}$ a $\mathbf{V}$, popř. možnostmi určení jedné z těchto matic pomocí druhé. Ponecháváme jako snadné cvičení formulaci algoritmů vzájemného převodu mezi maticemi $\mathbf{A}$ a $\mathbf{V}$. Při přechodu od $\mathbf{V}$ k $\mathbf{A}$ musíme pochopitelně zvolit číslování hran, takže $\mathbf{A}$ bude určena až na pořadí sloupců. Dále je

třeba brát v úvahu, že při vyjádření grafu maticí **A** se nepřipouštějí smyčky, které lze tolerovat při použití matice **V**. Následující algebraický vztah mezi maticemi **A** a **V** není vhodný jako návod pro převod **A** na **V** a zcela nepoužitelný při převodu **V** na **A**!

Věta 4.7: Nechť **A** je matice incidence a **V** matice sousednosti neorientovaného grafu $G = \langle H, U, \varrho \rangle$, nechť $\mathbf{D} = [d_{ii}]$ je diagonální matice s prvky $d_{ii} = \delta_G(u_i)$. Potom platí

$$\mathbf{A} \cdot \mathbf{A}^T = \mathbf{V} + \mathbf{D}, \tag{4.9}$$

kde s maticí **A** na levé straně počítáme jako s celočíselnou.

Důkaz: Označme $\mathbf{A} \cdot \mathbf{A}^T = [w_{ik}]$, $|H| = m$, potom platí

$$w_{ik} = \sum_{j=1}^{m} a_{ij} a_{jk}^T = \sum_{j=1}^{m} a_{ij} a_{kj}.$$

Součin $a_{ij}a_{kj}$ má pro $i \neq k$ hodnotu 1 právě tehdy, když hrana h_j inciduje s uzly u_i, u_k, v ostatních případech je roven 0. Sečtením přes všechny indexy j (všechny hrany) tedy dostaneme hodnotu v_{ik}. Pro $i = k$ dostáváme

$$w_{ik} = \sum_{j=1}^{m} a_{ij}^2 = \delta_G(u_i) = d_{ii}.$$

Permutací sloupců matice **A** se nezmění hodnota součtů, takže výsledná matice **V** nezáleží na očíslování množiny hran. △

Existují další druhy matic, které se používají k vyjádření struktury grafu (viz např. [29]). Jak však naznačil příklad na obr. 4.1, obsahuje matice **V** a především pak matice **A** pro grafy o větším počtu uzlů mnoho nulových prvků. Základní nevýhodou maticového vyjádření grafu pro potřeby počítačového zpracování jsou tedy nejen paměťové nároky vyplývající z velikosti matic **A** nebo **V**, ale především rozptýlený charakter informace, kterou je třeba používat při realizaci většiny operací a řešení typických úloh na grafech.

Paměťové nároky je sice možné snížit na únosnou mez využitím binární povahy ukládaných údajů, tím se však dále komplikuje přístup k jednotlivým hodnotám (zato operace např. s celými řádky lze realizovat poměrně efektivně). Pro zadávání struktury grafu formou vstupních dat je maticové vyjádření rovněž nevhodné – je nepřehledné a nedovoluje snadnou kontrolu správnosti zadání.

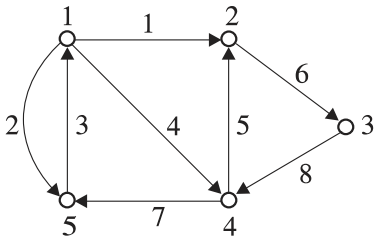
Matice orientovaných grafů

Podobně jako u neorientovaných grafů patří mezi základní formy vyjádření struktury orientovaných grafů matice reprezentující buď incidenci σ, nebo relaci následování Γ grafu. Podáme nyní definici těchto matic pro orientované grafy a ukážeme jejich vlastnosti.

Definice 4.8: Nechť $G = \langle H, U, \sigma \rangle$ je orientovaný graf s množinou hran $H = \{h_1, h_2, \ldots, h_m\}$ a množinou uzlů $U = \{u_1, u_2, \ldots, u_n\}$. **Maticí incidence** grafu G pak nazýváme celočíselnou matici $\mathbf{A} = [a_{ik}]$ typu (m, n), jejíž prvky jsou dány vztahem

$$a_{ik} = \begin{cases} 1, & \text{pokud } \sigma(h_k) = (u_i, u_j) \text{ pro jisté } u_j \in U, \\ -1, & \text{pokud } \sigma(h_k) = (u_j, u_i) \text{ pro jisté } u_j \in U, \\ 0, & \text{v ostatních případech} \end{cases} \tag{4.10}$$

Je vidět, že matice incidence orientovaného grafu je v podstatě matice incidence odpovídajícího neorientovaného grafu, v níž má jednička vyjadřující koncový uzel každé hrany přisvojeno záporné znaménko. Z pochopitelných důvodů nepřipouštíme ani u orientovaného grafu vyjadřovaného maticí incidence žádné smyčky. Příklad orientovaného grafu a jemu odpovídající matice incidence ukazujeme na obr. 4.2.



$A =$

	1	2	3	4	5	6	7	8
1	1	1	-1	1	0	0	0	0
2	-1	0	0	0	-1	1	0	0
3	0	0	0	0	0	-1	0	1
4	0	0	0	-1	1	0	1	-1
5	0	-1	1	0	0	0	-1	0

$V =$

	1	2	3	4	5
1	0	1	0	1	1
2	0	0	1	0	0
3	0	0	0	1	0
4	0	1	0	0	1
5	1	0	0	0	0

Obrázek 4.2: Matice incidence a sousednosti orientovaného grafu

Základní vlastnosti a způsob vytváření matice **A** jsou na první pohled zřejmé. Každý sloupec obsahuje jeden prvek 1 a jeden prvek -1, ostatní prvky jsou rovny nule, počet prvků 1 v i-tém řádku je roven $\delta_G^+(u_i)$, počet prvků -1 v i-tém řádku je roven $\delta_G^-(u_i)$.

Matice **A** orientovaného grafu má s ohledem na izomorfismus, souvislost grafu a lineární nezávislost řádků stejné vlastnosti, jako jsme uvedli u neorientovaných grafů (ponecháváme jako cvičení důkaz analogie věty 4.4 pro orientovaný graf). Velmi snadno se z matice **A** orientovaného grafu získá matice **A** neorientovaného grafu vzniklého zrušením orientace: místo a_{ik} použijeme $|a_{ik}|$.

Definice 4.9: Nechť $G = \langle H, U, \sigma \rangle$ je orientovaný graf s množinou uzlů $U = \{u_1, u_2, \ldots, u_n\}$. **Maticí sousednosti** grafu G nazýváme čtvercovou celočíselnou matici $\mathbf{V} = [v_{ik}]$ řádu n, jejíž prvky jsou dány vztahem

$$v_{ik} = |\sigma^{-1}(u_i, u_k)|. \tag{4.11}$$

Prvek v_{ik} tedy určuje počet orientovaných hran vedoucích z u_i do u_k. V případě prostého grafu je matice **V** maticovým vyjádřením relace následnosti Γ grafu G, relaci předcházení Γ^{-1} vyjadřuje transponovaná matice $\mathbf{V}^T$. Matice sousednosti orientovaného grafu není obecně (na rozdíl od neorientovaného případu) symetrická. U grafu z obr. 4.2 je rovněž uvedena jeho matice sousednosti **V**. Hodnoty diagonálních prvků matice opět vyjadřují případnou existenci smyček.

Přechod k neorientovanému grafu zrušením orientace vyjádříme zvětšením hodnot nediagonálních prvků o hodnotu symetricky umístěnou vůči hlavní diagonále, tzn.

$$v'_{ik} = v_{ik} + v_{ki} \quad \text{pro } k \neq i. \tag{4.12}$$

To je také možné chápat jako vytvoření symetricky orientovaného grafu.

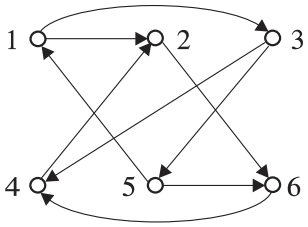
Při vhodném očíslování uzlů lze pro graf o p komponentách docílit tvaru (4.6) i pro matici **V** orientovaného grafu. Také tvrzení věty 4.6 bude platit s tím rozdílem, že prvek $v_{ik}^{(r)}$ matice $\mathbf{V}^r$ udává počet spojení délky r z uzlu u_i do uzlu u_k.

Při určování souvislosti a složení komponent orientovaného grafu pomocí matice **V** lze použít (neefektivní) postup uvedený v předchozím odstavci (t.j. výpočtem matice dosažitelnosti) pro matici symetrizace daného grafu, tzn. pro $\mathbf{V} + \mathbf{V}^T$. Popsaný postup lze ovšem využít i pro určení silných komponent. Spočítáme-li matici

$$\mathbf{V}^* = \sum_{i=0}^{d} V^i, \quad d = \min(|H|, |U| - 1), \tag{4.13}$$

pak tato matice svými nenulovými prvky v_{ik}^* určuje existenci spojení z uzlu u_i do uzlu u_j. Místo matice $\mathbf{V}^*$ lze však s použitím Booleových operací spočítat přímo matici dosažitelnosti **R** jako maticovou reprezentaci reflexivně-tranzitivního uzávěru relace následnosti Γ. Pro prvky této matice platí:

$$r_{ik} = \begin{cases} 1 & \text{pokud existuje spojení z uzlu } u_i \text{ do } u_k, \\ 0 & \text{v opačném případě.} \end{cases}$$



$\boldsymbol{R}$ =

	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	0	1	0	1
3	1	1	1	1	1	1
4	0	1	0	1	0	1
5	1	1	1	1	1	1
6	0	1	0	1	0	1

$\boldsymbol{R'}$ =

	1	2	3	4	5	6
1	1	0	1	0	1	0
2	0	1	0	1	0	1
3	1	0	1	0	1	0
4	0	1	0	1	0	1
5	1	0	1	0	1	0
6	0	1	0	1	0	1

Obrázek 4.3: Matice dosažitelnosti

Nyní stačí sestrojit matici $\mathbf{R}' = \mathbf{R} \wedge \mathbf{R}^T$ s prvky $r'_{ik} = \min(r_{ik}, r_{ki})$, jejíž hodnoty v i-tém řádku (sloupci) určují uzly náležející téže silné komponentě grafu G.

Pro graf na obr. 4.3 můžeme získat strukturu silných komponent následujícím postupem. Nejprve určíme uzly dosažitelné z jednotlivých uzlů grafu:

$$\begin{aligned}
\Gamma^*(1) &= \Gamma^0(1) \cup \Gamma^1(1) \cup \Gamma^2(1) \ldots = \\
&= \{1\} \cup \{2,3\} \cup \{4,5,6\} = \{1,2,3,4,5,6\}, \\
\Gamma^*(2) &= \{2\} \cup \{6\} \cup \{4\} = \{2,4,6\}, \\
\Gamma^*(3) &= \{3\} \cup \{4,5\} \cup \{1,2,6\} = \{1,2,3,4,5,6\}, \\
\Gamma^*(4) &= \{4\} \cup \{2\} \cup \{6\} = \{2,4,6\}, \\
\Gamma^*(5) &= \{5\} \cup \{1,6\} \cup \{2,3,4\} = \{1,2,3,4,5,6\}, \\
\Gamma^*(6) &= \{6\} \cup \{4\} \cup \{2\} = \{2,4,6\}.
\end{aligned}$$

Nyní sestavíme matici dosažitelnosti $\mathbf{R}$ uvedenou na obr. 4.3 a konečně matici $\mathbf{R}' = \mathbf{R} \wedge \mathbf{R}^T$. Z prvního řádku matice $\mathbf{R}'$ vidíme, že jedna silná komponenta našeho grafu obsahuje uzly $\{1,3,5\}$, a z druhého řádku je vidět, že zbývající uzly $\{2,4,6\}$ tvoří druhou silnou komponentu.

O tom, že maticové vyjádření struktury grafu nedovoluje zpravidla realizovat efektivně grafové algoritmy, jsme se již zmínili. Platí to např. i pro použití matice $\mathbf{V}$ při testování acykličnosti orientovaného grafu pomocí následující algebraické charakterizace.

Věta 4.10: Nechť $\mathbf{V}$ je matice sousednosti orientovaného grafu G a $\mathbf{E}$ je jednotková matice téhož řádu jako $\mathbf{V}$. Potom je graf G acyklický právě tehdy, je-li matice $\mathbf{E} - \mathbf{V}$ regulární.

Důkaz: Z lineární algebry je známo, že pro regulární matici $\mathbf{E} - \mathbf{V}$ platí

$$(\mathbf{E} - \mathbf{V})^{-1} = \mathbf{E} + \mathbf{V} + \mathbf{V}^2 + \ldots + \mathbf{V}^k \ldots \tag{4.14}$$

Řada celočíselných matic na pravé straně však bude konvergovat právě tehdy, platí-li pro nějaké $k \in \mathbf{N}$ rovnost $\mathbf{V}^k = 0$. To ale znamená, že v daném grafu neexistují spojení délky k a větší, takže nemůže obsahovat cyklus. △

Algoritmus vzájemného převodu mezi maticemi sousednosti a incidence ponecháme opět jako cvičení. Algebraický vztah mezi těmito maticemi ukazuje následující tvrzení.

Věta 4.11: Nechť $\mathbf{A}$ je matice incidence a $\mathbf{V}$ matice sousednosti orientovaného grafu G. Nechť $\mathbf{D} = [d_{ii}]$ je diagonální matice s prvky $d_{ii} = \delta_G^+(u_i) + \delta_G^-(u_i)$. Potom platí

$$\mathbf{A} \cdot \mathbf{A}^T = \mathbf{D} - \mathbf{V} - \mathbf{V}^T \tag{4.15}$$

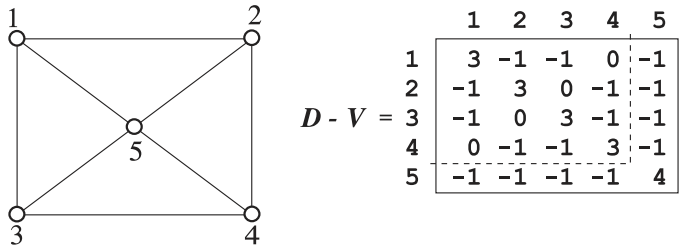
Důkaz: Důkaz tohoto vztahu je obdobný jako u věty 4.7. △

Studium vlastností matic orientovaných i neorientovaných grafů dává mnoho zajímavých a užitečných výsledků. Některé z těchto výsledků lze nalézt např. v [21] nebo [9], v tomto textu

uvedeme bez důkazu jen následující tvrzení, které umožňuje určit algebraicky počet různých koster grafu.

Věta 4.12: Nechť $\mathbf{A}_r$ je matice tvořená libovolnými $|U|-1$ lineárně nezávislými řádky incidenční matice $\mathbf{A}$ souvislého orientovaného grafu $G = \langle H, U, \sigma \rangle$. Počet různých koster grafu G je pak dán výrazem

$$\det(\mathbf{A}_r \mathbf{A}_r^T). \tag{4.16}$$



Obrázek 4.4: Určení počtu koster grafu

Pomocí výrazu (4.16) můžeme samozřejmě určit rovněž počet koster neorientovaného grafu. Stačí, když si jej představíme libovolným způsobem orientovaný, určíme pak příslušnou matici $\mathbf{A}_r \mathbf{A}_r^T$ a vypočteme její determinant. Matici $\mathbf{A}_r \mathbf{A}_r^T$ je však možné také určit přímo z matice sousednosti $\mathbf{V}$ neorientovaného grafu tak, že vybereme hlavní čtvercovou podmatici řádu $|U|-1$ matice $\mathbf{D}-\mathbf{V}$, kde $\mathbf{D}$ je diagonální matice stupňů uzlů.

Uvedeným postupem použitým na graf z obr. 4.4 dostaneme matici $\mathbf{D} - \mathbf{V}$ znázorněnou na stejném obrázku. Determinant vybrané podmatice je 45, takže počet různých koster i pro tento poměrně jednoduchý graf je příliš velký na to, abychom se je pokoušeli všechny znázornit. Z výrazu (4.16) dostaneme jednoduchou algebraickou úpravou počet koster úplného grafu o n uzlech, který je roven

$$n^{n-2} \tag{4.17}$$

Je třeba připomenout, že výrazy (4.16) a (4.17) určují **počty všech koster**, mezi nimiž je velmi mnoho koster vzájemně izomorfních.

Spojová reprezentace grafu

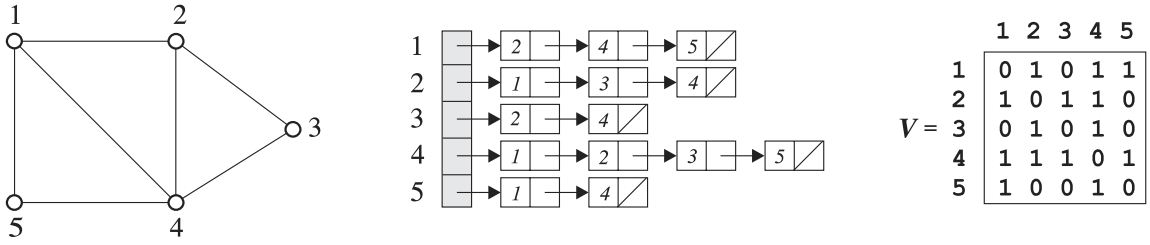
Jako nevýhody maticové reprezentace grafu jsme uvedli paměťové nároky a rozptýlenost uložené informace. Z programovacích technik je dobře známa spojová metoda ukládání řídkých matic – jejími variantami jsou i spojové způsoby vyjadřování struktury grafu.

Základní forma spojové reprezentace neorientovaného grafu $G = \langle H, U, \varrho \rangle$ sestává z pole Adj obsahujícího $|U|$ odkazů (ukazatelů) na seznamy sousedů jednotlivých uzlů z množiny U. Pro každý uzel $u \in U$ obsahuje spojový seznam $Adj[u]$ jeden záznam pro každou neorientovanou hranu $h = [u, v] \in H$ s druhým krajním uzlem v. Záznam typicky obsahuje identifikaci (číslo) uzlu v, případně identifikaci nebo doplňující údaje týkající se hrany h, pořadí záznamů je obecně libovolné.

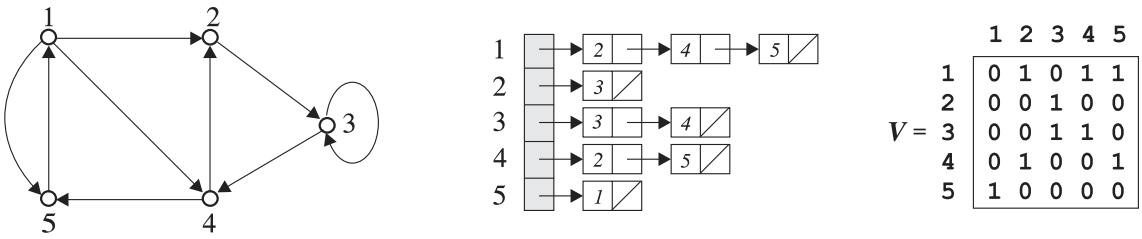
Na obr. 4.5 ukazujeme neorientovaný graf společně s jeho možnou spojovou reprezentací. Je vidět, že paměťová složitost spojové reprezentace je dána součtem délky pole ukazatelů (t.j. $|U|$) a celkovou délkou všech seznamů sousedů (t.j. $2|H|$). V asymptotickém vyjádření to představuje

$$O(|U| + 2|H|) = O(|U| + |H|) = O(\max(|U|, |H|))$$

Při přechodu k orientovaným grafům se na spojové reprezentaci téměř nic nezmění, záznamy ve spojovém seznamu však odpovídají orientovaným hranám vycházejícím z daného uzlu. To znamená, že pro každý uzel $u \in U$ obsahuje spojový seznam $Adj[u]$ jeden záznam pro každou orientovanou hranu $h = (u, v) \in H$ s koncovým uzlem v. Tento záznam může vedle identifikace uzlu v opět obsahovat i další údaje, např. délku hrany h pro hranově ohodnocené grafy, atd.



Obrázek 4.5: Spojová reprezentace neorientovaného grafu



Obrázek 4.6: Spojová reprezentace orientovaného grafu

Možnou spojovou reprezentaci orientovaného grafu ukazujeme na obr. 4.6. Každá hrana se nyní v seznamu následníků projeví právě jednou, což sice znamená menší skutečné paměťové nároky reprezentace, ale její asymptotická paměťová složitost bude stále $O(|U| + |H|)$ jako pro neorientované grafy.

Cvičení

4.1-1. S uvážením skutečných minimálních paměťových nároků na uložení jednotlivých typů údajů používaných pro vyjádření struktury grafu určete pro hodnoty $|U| = 10, 100, 1000, 10000$, při jakém počtu hran je spojová reprezentace grafu paměťově výhodnější. Uvažujte odděleně neorientované a orientované grafy.

4.1-2. Nechť T je úplný pravidelný strom stupně 2 se sedmi uzly, jejichž očíslování odpovídá jejich pořadí v binární haldě. Vytvořte matici incidence, matici sousednosti a spojovou reprezentaci grafu T chápaného jako neorientovaný graf. Vytvořte rovněž matici incidence, matici sousednosti a spojovou reprezentaci (orientovaného) kořenového stromu T.

4.1-3. Určete asymptotickou výpočetní složitost výpočtu vstupního, resp. výstupního stupně všech uzlů orientovaného grafu na základě jeho

(a) matice incidence **A** (b) matice sousednosti **V** (c) spojové reprezentace.

4.1-4. Navrhněte efektivní algoritmus pro vytvoření opačně orientovaného grafu G^- k orientovanému grafu G zadanému

(a) maticí incidence **A** (b) maticí sousednosti **V** (c) spojovou reprezentací.

Výsledný graf má být vždy vyjádřen stejnou formou jako graf výchozí. Určete rovněž časovou složitost navrženého algoritmu.

4.1-5. Navrhněte algoritmus s časovou složitostí $O(|U| + |H|)$, který ze spojové reprezentace neorientovaného multigrafu G odvodí spojovou reprezentaci obyčejného grafu G', v němž nebudou smyčky a každá skupina rovnoběžných hran bude nahrazena hranou jedinou.

4.1-6. Navrhněte efektivní algoritmus pro vytvoření druhé mocniny G^2 (t.j. složení $G \circ G$) orientovaného grafu G zadaného

(a) maticí incidence **A** (b) maticí sousednosti **V** (c) spojovou reprezentací.

Výsledný graf má být vždy vyjádřen stejnou formou jako graf výchozí. Určete rovněž časovou složitost navrženého algoritmu.

4.1-7. Při použití maticové reprezentace bývá časová složitost většiny grafových algoritmů alespoň $\Theta(|V|^2)$, existují však výjimky. Navrhněte algoritmus, který pro graf zadaný maticí sousednosti $\mathbf{V}$ určí v čase $O(|U|)$, zda daný graf obsahuje **stok**, t.j. uzel s nulovým výstupním stupněm a vstupním stupněm rovným $|U| - 1$.

4.1-8. Dokažte větu 4.11.

4.1-9. Zdůvodněte, proč ve vztahu (4.13) stačí sčítat do uvedené hodnoty d.

4.1-10. Nechť $\mathbf{V}$ je matice sousednosti úplného (neorientovaného) grafu K_5.

(a) Zdůvodněte, proč jsou v matici $\mathbf{V}^n$ všechny diagonální a mimodiagonální prvky stejné (označme jejich hodnoty jako d_n, resp. v_n).

(b) Zdůvodněte platnost následujících rekurentních vztahů:

$$d_{n+1} = 4 \cdot a_n, \quad a_{n+1} = d_n + 3 \cdot a_n, \quad a_{n+1} = 3 \cdot a_n + 4 \cdot a_{n-1}$$

a vyjádřete hodnoty d_n a a_n v uzavřeném tvaru.

(c) Odvoďte vztahy pro hodnoty diagonálních a mimodiagonálních prvků n-té mocniny $\mathbf{V}^n$ matice sousednosti úplného grafu K_m.

4.1-11. Nechť $\mathbf{V}$ je matice sousednosti úplného bipartitního grafu $K_{m,n}$. Nalezněte výrazy určující hodnoty prvků k-té mocniny $\mathbf{V}^k$ této matice.

4.2 Prohledávání do šířky

Prohledání grafu, t.j. systematická prohlídka všech jeho uzlů i hran, patří mezi nejjednodušší a současně nejpotřebnější algoritmické postupy na grafech. **Prohledávání do šířky** (angl. **breadth-first search**) představuje základní variantu postupu prohledávání a z jeho hlavních myšlenek vycházejí další důležité grafové algoritmy – např. Dijkstrův algoritmus hledání nejkratší cesty (viz odst. 6.2) nebo Jarníkův-Primův algoritmus hledání minimální kostry grafu (viz odst. 5.4). V tomto odstavci budeme bez újmy na obecnosti předpokládat pouze obyčejné grafy (jak neorientované tak i orientované).

Pro zadaný graf $G = \langle H, U \rangle$ a vyznačený uzel s se prostřednictvím prohledávání do šířky systematicky prověřují hrany grafu G s cílem „nalézt“ každý uzel, který je dosažitelný z uzlu s. Prohledáváním do šířky se současně stanoví vzdálenost od s ke každému dosažitelnému uzlu, a jako vedlejší produkt se tak získá **strom prohledávání do šířky** obsahující všechny z s dosažitelné uzly. Pro každý uzel v obsažený v tomto stromu – nadále jej pro stručnost budeme označovat jako **BF-strom** – je cesta z kořene s do uzlu v zároveň nejkratší cestou z s do v v grafu G. BF-strom je tedy současně **stromem nejkratších cest** z uzlu s do všech dosažitelných uzlů. V popsané podobě funguje algoritmus prohledávání do šířky pro neorientované i pro orientované grafy.

Název tohoto způsobu prohledávání vyjadřuje uniformitu postupu, při kterém se hranice mezi objevenými a (dosud) neobjevenými uzly rovnoměrně posouvá na celé své šířce. To má za následek, že algoritmus objeví všechny uzly ležící ve vzdálenosti k od uzlu s dříve, než první uzel ve vzdálenosti $k + 1$. V průběhu prohledávání je každý algoritmem zpracovávaný uzel označen nejprve jako FRESH, potom jako OPEN a nakonec jako CLOSED. Na začátku jsou všechny uzly **nové** (FRESH). Jakmile algoritmus poprvé narazí při prohledávání na určitý uzel (t.j. „objeví“ jej), označí jej jako **otevřený** (OPEN). Jako **uzavřený** (CLOSED) pak označí každý uzel, který už nepatří do hranice mezi objevenými a neobjevenými uzly – tedy uzel, který už zaručeně nemá žádného nového souseda.

BF-strom obsahuje na počátku pouze svůj kořen, kterým je uzel s. Jakmile se v průběhu procházení seznamu sousedů (následníků) nějakého uzlu u objeví nový uzel v, doplní se BF-strom o hranu (u, v) a uzel v. Uzel u se tak stane **předchůdcem (rodičem)** uzlu v v BF-stromu. Každý uzel může být objeven nejvýše jednou, takže může mít také nejvýše jednoho předchůdce. Vedle základních relací vyjádřených pojmy předchůdce a následník budeme používat také reflexivně-tranzitivní uzávěry těchto relací pojmenované jako **předek** a **potomek**. Je tedy každý uzel y nacházející se na cestě z s do x (včetně uzlu x samotného) předkem uzlu x a naopak uzel x je potomkem každého takového uzlu y. Je důležité nezapomínat, že všechny zmiňované relace jsou odvozeny podle struktury BF-stromu, nikoliv podle výchozího grafu, takže relace „být předkem“ nebo „být potomkem“ mají charakter uspořádání.

V proceduře prohledávání do šířky BFS uvedené níže předpokládáme, že vstupní (obyčejný) graf $G = \langle H, U \rangle$ je zadán pomocí spojové reprezentace. Algoritmus hledání dále používá několika pomocných údajů vztahujících se ke každému uzlu a frontu, do níž ukládá otevřené uzly. Stav každého uzlu u (tj. FRESH, OPEN, CLOSED) zachycuje proměnná $stav[u]$, předchůdce uzlu u je uložen v proměnné $p[u]$. Pokud uzel u nemá žádného předchůdce (je to např. uzel s nebo dosud neobjevený uzel), pak je $p[u] =$ NIL. Vzdálenost od počátku s k uzlu u počítá algoritmus v proměnné $d[u]$.

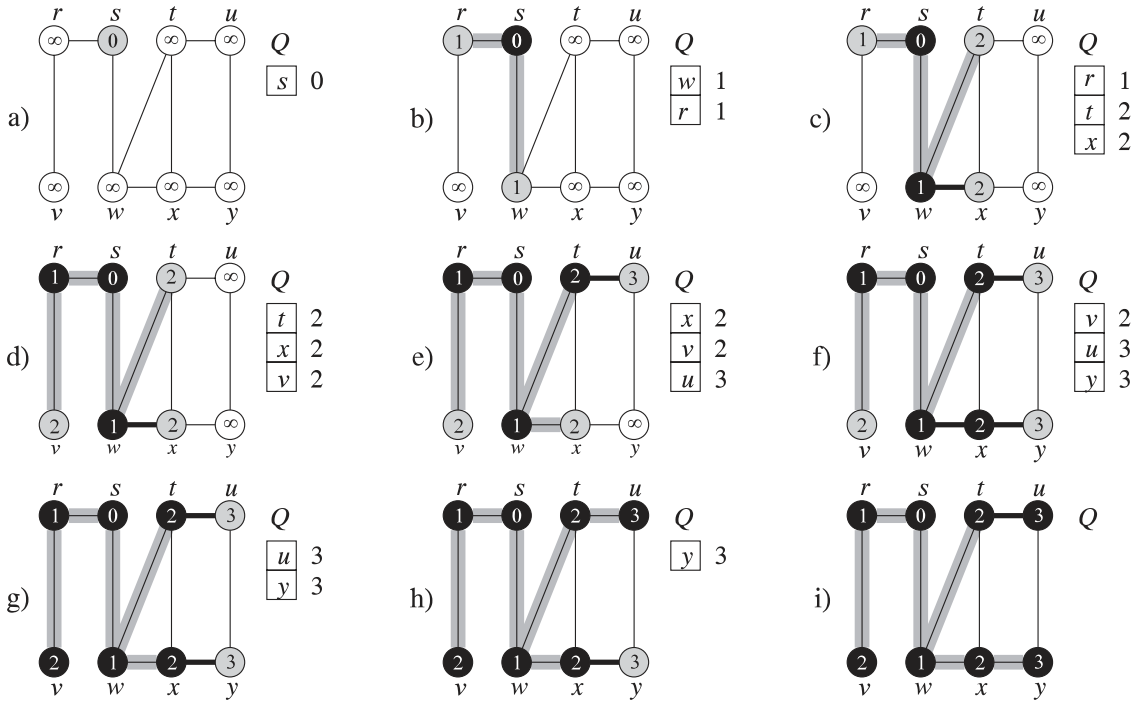
Algoritmus 4.13 Prohledávání do šířky

BFS(G, s)

```
1   for každý uzel u ∈ U − {s}          Všechny uzly mimo s
2       do stav[u] := FRESH             označ jako nové,
3          d[u] := ∞                    zatím nedosažitelné ze s
4          p[u] := NIL                  a bez předchůdce.
5   stav[s] := OPEN                     Samotný uzel s otevři
6   d[s] := 0                           s nulovou vzdáleností od s,
7   p[s] := NIL                         rovněž bez předchůdce
8   INIT_QUEUE; ENQUEUE(s)              a ulož jej do fronty.
9   while not EMPTY_QUEUE               Dokud fronta není prázdná,
10      do u := QUEUE_FIRST             vyber z ní první otevřený uzel
11         for každé v ∈ Adj[u] do      a jeho sousedy,
12            if  stav[v] = FRESH       kteří jsou noví,
13               then stav[v] := OPEN   otevři,
14                    d[v] := d[u] + 1  urči jim vzdálenost
15                    p[v] := u         i předchůdce
16                    ENQUEUE(v)        a ulož je do fronty.
17         DEQUEUE                      Odeber uzel u z fronty
18         stav[u] := CLOSED            a zavři jej.
```

Průběh prohledávání do šířky na neorientovaném grafu ukazuje obr. 4.7. Stav uzlu vyznačujeme stupněm šedi v jeho znázornění (bílá – FRESH, černá – CLOSED), stromové hrany se kreslí tučně. Číslo uvnitř uzlu v udává hodnotu $d[v]$. Obrázek zachycuje stavy vždy na začátku cyklu tvořeného řádky 9 až 18, vedle uzlů uložených ve frontě je pro zdůraznění znovu zopakována jejich hodnota vzdálenosti $d[v]$.

Věnujme se nejprve analýze časové složitosti algoritmu BFS pro graf $G = \langle H, U \rangle$. Kromě počáteční inicializace se nikdy nemůže stav uzlu změnit na FRESH, takže test na řádce 12 zaručuje, že se žádný uzel nemůže uložit do (a tedy ani vybrat z) fronty více než jednou. Operace ukládání a vybírání mají pro frontu složitost $O(1)$, takže celkový čas operací s frontou je $O(|U|)$. Seznam sousedů uzlu se prochází pouze při vybírání uzlu z fronty, a to nastává pro každý uzel nejvýše jednou. Protože celková délka seznamů sousedů pro všechny uzly je $\Theta(|H|)$, spotřebuje se procházením seznamů sousedů celkově nejvýše $O(|H|)$. Inicializační část má ovšem složitost $O(|U|)$, takže celková složitost bude $O(|U|+|H|)$. Časová složitost algoritmu BFS je tedy stejná jako paměťová složitost spojové reprezentace grafu.



Obrázek 4.7: Prohledávání do šířky

Nyní se budeme věnovat algoritmu BFS s ohledem na vlastnosti jím vytvořeného BF-stromu a spočtených hodnot $d[u]$ pro uzly dosažitelné z uzlu s. Měli bychom potvrdit již dříve uvedené prohlášení, že pro každý (z uzlu s dosažitelný) uzel u je hodnota $d[u]$ rovna vzdálenosti $d(s,u)$, a BF-strom je tedy stromem minimálních cest z uzlu s do všech dosažitelných uzlů. Uvedeme nejprve několik pomocných tvrzení.

Lemma 4.14: Nechť $G = \langle H, U\rangle$ je prostý neorientovaný nebo orientovaný graf a $s \in U$ je libovolný jeho uzel. Potom pro každou hranu $(u,v) \in H$ platí

$$d(s,v) \leq d(s,u) + 1. \tag{4.18}$$

Důkaz: Není-li uzel u dosažitelný z uzlu s, potom je $d(s,u) = \infty$ a uvedená nerovnost platí bez ohledu na to, zda je uzel v dosažitelný. Pro dosažitelný uzel u nemůže být nejkratší cesta z s do v delší než nejkratší cesta z s do u prodloužená o hranu (u,v). △

Lemma 4.15: Nechť $G = \langle H, U\rangle$ je prostý neorientovaný nebo orientovaný graf a předpokládejme, že na G provedeme prohledání pomocí algoritmu BFS vycházející z uzlu $s \in U$. Potom po ukončení BFS bude pro vypočtené hodnoty $d[v]$ platit nerovnost $d[v] \geq d(s,v)$ pro každý uzel $v \in U$.

Důkaz: (indukcí podle počtu uzlů uložených do fronty)
1) V okamžiku, kdy je do fronty uložen první uzel, kterým je uzel s na řádce 8 algoritmu BFS, platí $d[s] = 0 = d(s,s)$ a $d[v] = \infty \geq d(s,v)$ pro $v \in U - \{s\}$. Základ indukce je tedy splněn.

2) Mějme nyní uzel v ukládaný do fronty jako $(n+1)$-ní v pořadí, pro všechny uzly před ním pokládáme naši indukční hypotézu za splněnou. Označme jako u uzel, v jehož seznamu sousedů byl uzel v poprvé objeven algoritmem BFS. Uzel u byl jistě do fronty uložen někdy dříve, takže podle indukčního předpokladu je $d[u] \geq d(s,u)$. Ze způsobu výpočtu hodnoty $d[v]$ na řádce 14 a ze vztahu (4.18) tak dostáváme

$$d[v] = d[u] + 1 \geq d(s,u) + 1 \geq d(s,v).$$

Po tomto výpočtu hodnoty $d[v]$ je uzel v označen jako OPEN a vzápětí uložen do fronty, takže už do ní nemůže být nikdy uložen znovu, neboť řádky 13 až 16 se provádějí pouze pro nové uzly. Hodnota $d[v]$ se tedy již nezmění a dokazované tvrzení platí. △

Lemma 4.16: Nechť při provádění algoritmu BFS na grafu $G = \langle H, U\rangle$ obsahuje fronta posloupnost uzlů $\langle v_1, v_2, \ldots, v_r\rangle$, přičemž v_1 je první a v_r poslední uzel ve frontě. Potom platí

$$d[v_r] \leq d[v_1] + 1 \quad \& \quad d[v_i] \leq d[v_{i+1}] \text{ pro } i = 1, 2, \ldots, r-1. \tag{4.19}$$

Důkaz: (indukcí podle počtu operací ENQUEUE / DEQUEUE s frontou)
1) Po provedení první operace se uloží do fronty uzel s, takže tvrzení zřejmě platí.

2) Nechť platí indukční hypotéza pro $n(\geq 1)$ operací s frontou, ukážeme, že bude platit i pro $(n+1)$–ní operaci, ať je to DEQUEUE nebo ENQUEUE. Uvažujme nejprve operaci DEQUEUE, kterou se vybere první uzel v_1 z fronty. Je–li po jeho vybrání fronta prázdná, tvrzení je triviálně splněno, uvažujme tedy případ, že novým prvkem na začátku fronty se stane uzel v_2. Potom podle indukčního předpokladu je $d[v_r] \leq d[v_1] + 1 \leq d[v_2] + 1$ a ostatní nerovnosti v našem tvrzení zůstávají v platnosti, takže pro výběr je indukční hypotéza potvrzena.
Uvažujme nyní operaci ENQUEUE, k níž dochází (kromě prvního případu) pouze na řádce 16. V tomto okamžiku se uzel v stává uzlem v_{r+1} díky tomu, že byl objeven jako následník aktuálně zpracovávaného uzlu u, což je první uzel fronty v_1, který má být právě z fronty vybrán. Platí tedy $d[v_{r+1}] = d[v] = d[u] + 1 = d[v_1] + 1$. Současně ovšem máme $d[v_r] \leq d[v_1] + 1 = d[u] + 1 = d[v] = d[v_{r+1}]$ a zbývající nerovnosti zůstávají zachovány. Tvrzení tedy platí i pro případ ukládání do fronty. △

Vytvořili jsme tedy potřebné předpoklady pro důkaz správnosti algoritmu BFS ohledně nalezení stromu nekratších cest.

Věta 4.17: Nechť $G = \langle H, U\rangle$ je neorientovaný nebo orientovaný graf, na němž se provede prohledání pomocí algoritmu BFS z počátečního uzlu $s \in U$. Potom BFS nalezne všechny uzly $v \in U$ dosažitelné z uzlu s a po skončení bude $d[v] = d(s, v)$ pro každé $v \in U$. Kromě toho bude pro každý dosažitelný uzel $v \neq s$ nalezena nějaká nejkratší cesta z s do v taková, že ji tvoří nekratší cesta z s do $p[v]$ prodloužená o hranu $(p[v], v)$.

Důkaz: Řádka 14 algoritmu BFS se provádí pouze pro objevené – a tedy z s dosažitelné – uzly, takže se jenom pro ně počáteční nekonečná hodnota $d[v]$ změní na konečnou. Nedosažitelným uzlům zůstane hodnota $d[v] = \infty = d(s, v)$. Věnujme se tedy důkazu tvrzení pro uzly dosažitelné z uzlu s.

Označme jako U_k množinu uzlů ležících ve vzdálenosti k od uzlu s, tedy $U_k = \{v \in U : d(s, v) = k\}$. Důkaz provedeme indukcí podle k a naší indukční hypotézou bude, že se pro každý uzel $v \in U_k$ uskuteční během provádění BFS právě jedenkrát následující skupina operací:

- uzel v je označen jako OPEN
- $d[v]$ je nastaveno na hodnotu k
- je-li $v \neq s$, pak se $p[v]$ přiřadí hodnota u pro jisté $u \in U_{k-1}$
- uzel v se uloží do fronty

Uvedené operace se zjevně pro každý uzel mohou provést nejvýše jednou.

1) Pro $k = 0$ máme $U_0 = \{s\}$, neboť s je jediný uzel ve vzdálenosti 0 od s. Během inicializace na řádcích 5 až 8 se provedou požadované operace, takže indukční hypotéza platí.

2) V indukčním kroku využijeme toho, že po uložení uzlu u do fronty se již nezmění jeho hodnoty $d[u]$ a $p[u]$, a dále že před ukončením algoritmu nemůže být fronta nikdy prázdná. Ukládají-li se tedy uzly do fronty v pořadí $v_1, v_2, \ldots, v_r$, bude podle lemmatu 4.15 odpovídající posloupnost hodnot vzdáleností neklesající: $d[v_i] \leq d[v_{i+1}]$ pro $i = 1, 2, \ldots, r-1$.

Uvažujme nyní libovolný uzel $v \in U_k$ pro $k \geq 1$. Z monotónnosti posloupnosti hodnot vzdáleností, z indukčního předpokladu a z platnosti $d[v] \geq d(s, v)$ (viz lemma 4.15) vyplývá, že uzel v bude objeven teprve poté, co byly do fronty uloženy všechny uzly z množiny U_{k-1}.

Protože je $d(s,v) = k$, musí existovat cesta délky k z s do v, jejíž poslední hrana $(u,v) \in H$ má uzel u z množiny U_{k-1}. Bez újmy na obecnosti můžeme předpokládat, že uzel u byl otevřen jako první z těch uzlů množiny U_{k-1}, kreré jsou současně sousedy uzlu v. Potom ale již byl uzel u uložen do fronty a časem se dostal na její začátek. Při jeho výběru z fronty (řádka 10) a procházení jeho seznamu sousedů (řádka 11) bude objeven uzel v. Nemohl být totiž objeven dříve jako soused nějakého uzlu z množiny U_j pro $j < k-1$, neboť pak by jeho vzdálenost od s nemohla být k, a uzel u je jeho prvním v pořadí sousedem z množiny U_{k-1}. V okamžiku jeho objevení se ale na řádkách 13 až 16 provedou právě všechny požadované operace, přičemž spočtená hodnota $d[v]$ je podle indukčního předpokladu rovna $d[u]+1 = (k-1)+1 = k$. Tím se potvrzuje indukční hypotéza i pro množinu U_k. △

Způsob, jakým se v algoritmu BFS zachycuje struktura BF-stromu je velmi úsporný: každý uzel v má prostřednictvím své hodnoty $p[v]$ určeného svého (jediného) předchůdce, případně příznak, že žádného předchůdce nemá ($p[v]$ =NIL). Toto vyjádření struktury je však možné pouze pro omezenou třídu grafů a navíc umožňuje efektivně realizovat jen některé operace.

Definice 4.18: Nechť $G = \langle H, U\rangle$ je neorientovaný nebo orientovaný graf. **P-podlesem** G_p grafu G určeným pomocí pole předchůdců p nazýváme takový faktor grafu, jehož každá komponenta je (kořenovým) stromem a jehož množina hran H_p je dána vztahem

$$H_p = \{(p[v], v) \in H : p[v] \neq \text{ NIL}\},$$

přičemž množina uzlů $S = \{v \in U : p[v] = \text{NIL}\}$ představuje právě všechny kořeny stromů obsažených v G_p. Je-li navíc $S = \{s\}$, nazýváme G_p **p-podstromem** grafu G s kořenem s.

Z tvrzení popisujících vlastnosti prohledávání do šířky algoritmem BFS bezprostředně vyplývá následující lemma, které charakterizuje algoritmem vytvořený BF-strom pomocí právě zavedeného pojmu p-stromu.

Lemma 4.19: Použitím algoritmu BFS vycházejícího z uzlu s grafu $G = \langle H, U\rangle$ se prostřednictvím pole p získá p-podstrom $G_p = \langle H_p, \Gamma^*(s)\rangle$ podgrafu indukovaného množinou uzlů dosažitelných z uzlu s.

Odkazy na předchůdce v poli $p[v]$ vyjadřují sice strukturu BF-stromu, jejich základní smysl je však určovat předchůdce uzlu v na nejkratší cestě z uzlu s do v. Celou posloupnost uzlů vyjadřující tuto nejkratší cestu získáme následujícím jednoduchým algoritmem.

Algoritmus 4.20 Určení cesty v p-stromu

CESTA(G, s, v)

```
1    if v = s                                        Cesta končí aniž začala
2       then write(s)                                výstup počátečního uzlu
3       else if p[v] =NIL                            v nemá předchůdce?
4               then write('cesta z ',s,' do ',v, 'neexistuje')
5               else CESTA(G, s, p[v])               Napřed cesta k předchůdci
6                    write(v)                        a pak koncový uzel v.
```

Cvičení

4.2-1. Ukažte výsledky prohledání grafu z obr. 4.7 do šířky při použití postupně každého z uzlů tohoto grafu jako výchozího uzlu.

4.2-2. Upravte algoritmus BFS tak, aby vycházel z reprezentace grafu incidenční maticí, resp. maticí sousednosti. Určete složitost těchto variant prohlížení grafu do šířky.

4.2-3. Zdůvodněte, proč hodnota $d[u]$ spočtená algoritmem BFS nezávisí na tom, v jakém pořadí jsou jednotlivé uzly uvedeny v seznamech sousedů. Je nějaká část výsledků algoritmu BFS závislá na tomto pořadí?

4.2-4. Uveďte příklad grafu $G = \langle H, U\rangle$, výchozího uzlu $s \in U$ a množiny hran $H_p \subseteq H$ p-stromu takových, že pro každý uzel $v \in U$ je (jediná) cesta z uzlu s do v ve stromu $G_p = \langle H_p, U\rangle$ současně nejkratší cestou v grafu G, a přitom p-strom G_p nelze získat provedením algoritmu BFS bez ohledu na to, jak by se uspořádaly uzly v seznamech sousedů.

4.2-5. Navrhněte efektivní algoritmus na zjištění toho, zda je daný neorientovaný graf bipartitní.
(*Návod:* Inspirujte se množinami U_k z důkazu věty 4.17 – graf nesmí obsahovat hrany spojující uzly z téže množiny U_k.)

4.2-6. Nechť $G = \langle H, U\rangle$ je souvislý neorientovaný graf. Navrhněte algoritmus složitosti $O(|U|+|H|)$, který nalezne sled grafu G procházející každou hranou právě jednou v obou možných směrech. Na základě tohoto algoritmu popište, jak lze najít cestu z bludiště, máme-li k dispozici dostatečné množství značkovacích žetonů.

4.3 Prohledávání do hloubky

Jak naznačuje název tohoto způsobu prohledávání, jeho základní strategie spočívá ve snaze postupovat co „nejhlouběji“ do grafu, jak je to jen možné. Algoritmus **prohledávání do hloubky** (angl. **depth-first search**, zkráceně **DFS**) probírá hrany vycházející z posledně nalezeného uzlu v, který má ještě nějaké neprobrané hrany. Když probere všechny jeho hrany, vrátí se zpátky k uzlu, z něhož se do uzlu v dostal, a z něho pokračuje po další dosud neprobrané hraně. Takovým způsobem se postupuje tak dlouho, dokud se neobjeví všechny uzly dosažitelné z prvního výchozího uzlu. Pokud zbývá nějaký neobjevený uzel, zvolí se jako další výchozí uzel, a prohledávání do hloubky pak pokračuje z něho. Tento postup se opakuje tak dlouho, až jsou objeveny všechny uzly.

Podobně jako při prohledávání do šířky uchovávají se i při prohledávání do hloubky o každém uzlu určité pomocné údaje, namísto fronty otevřených uzlů je zde ale (implicitní) zásobník zajišťující implementaci rekurze. Při objevení nového uzlu v jako následníka uzlu u se stejně jako u prohledávání do šířky přiřadí $p[v] := u$. Vzhledem k opakované volbě výchozího uzlu z dosud neobjevených uzlů je po skončení prohledávání do hloubky prostřednictvím odkazů $p[v]$ definován p-podles prohledávaného grafu tvořený obecně několika stromy. Tento p-podles budeme nazývat **DF-lesem** a každou jeho komponentu **DF-stromem**.

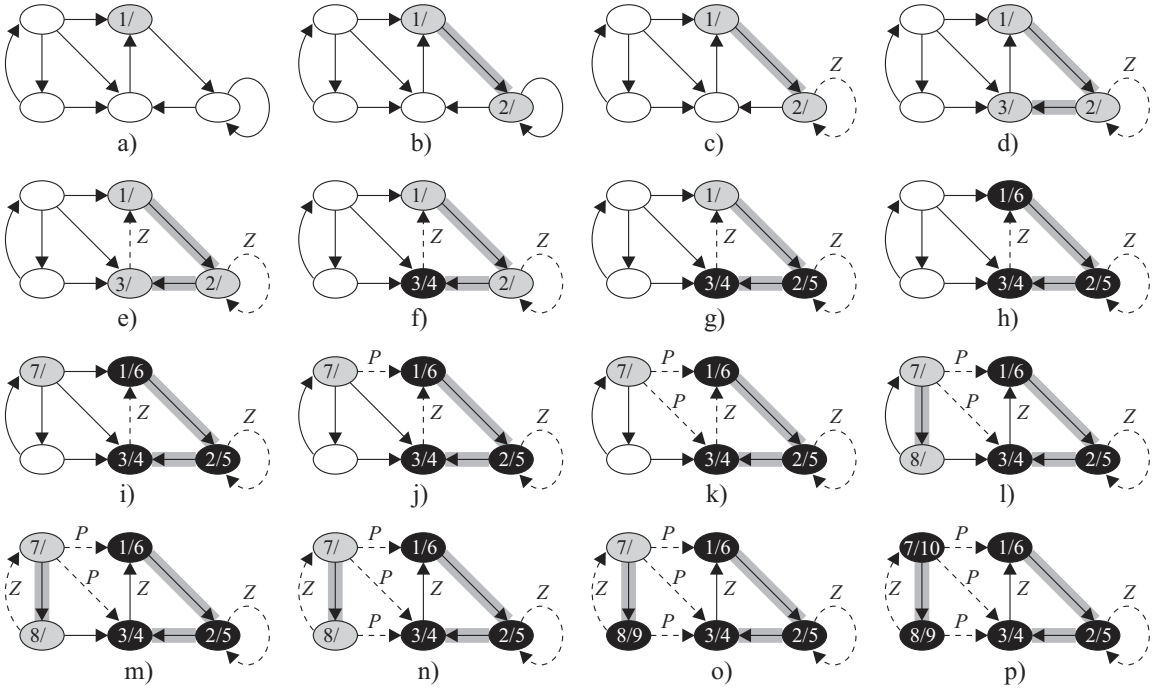
Algoritmus opět rozděluje uzly do tří skupin: nové (dosud neobjevené), otevřené a uzavřené. Otevřeným se uzel stane po svém prvním objevení, uzavřený je poté, co byl kompletně prozkoumán jeho seznam sousedů. Vedle toho se každému uzlu u přidělují dvě časové „značky“ uchovávané v poli $d[u]$ a $f[u]$: značka $d[u]$ odpovídá okamžiku objevení uzlu u (t.j. okamžiku jeho otevření), značka $f[u]$ odpovídá jeho uzavření (t.j. okamžiku skončení průchodu jeho seznamu sousedů). Tyto značky vypovídají nejen o průběhu prohledávání, ale odrážejí i řadu vlastností grafu, a tak se často používají v dalších grafových algoritmech. Hodnotami značek jsou přirozená čísla od 1 do $2|U|$, neboť pro každý uzel nastává právě jednou jeho objevení a právě jednou se uzavře. Po přidělení určité hodnoty se značkovací čítač zvýší o 1, a tato hodnota bude přidělena příště. Z tohoto způsobu přidělování značek plyne, že platí

$$d[u] < f[u] \quad \text{pro všechny uzly } u \in U. \tag{4.20}$$

Následující vyjádření algoritmu prohledávání do hloubky je použitelné jak pro neorientovaný tak pro orientovaný graf $G = \langle H, U\rangle$, předpokládá se ovšem spojová reprezentace grafu prostřednictvím seznamů sousedů (následníků). Na obr. 4.8 ukazujeme průběh provádění algoritmu při použití na orientovaný graf z obr. 4.6.

Algoritmus 4.21 Prohledávání do hloubky

DFS(G)



Obrázek 4.8: Prohledávání do hloubky

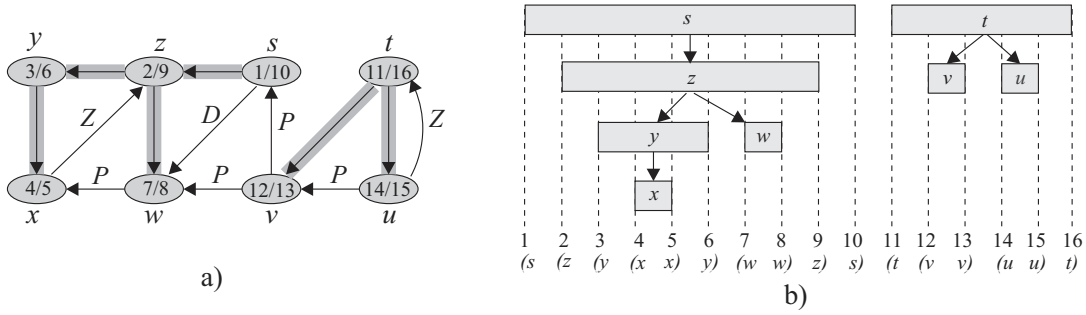
```
1     for každý uzel u ∈ U                    Počáteční nastavení
2         do stav[u] := FRESH                 všech uzlů na nové
3            p[u] := NIL                      a bez předchůdce.
4     i := 0                                  Nastavení čítače značek
5     for každý uzel u ∈ U                    Zkus všechny uzly
6         do if stav[u] = FRESH               a pro nové projdi
7               then DFS-Projdi(u)            jejich potomky do hloubky.

DFS-Projdi(u)
1     stav[u] := OPEN                         Prohlas uzel u za otevřený
2     i := i + 1; d[u] := i                   a přiděl mu první značku.
3     for každý uzel v ∈ Adj[u]               Projdi hranu (u, v)
4         do if stav[v] = FRESH               a pro nový uzel v
5               then p[v] := u                stanov za jeho předchůdce u
6                    DFS-Projdi(v)            a projdi potomky uzlu v.
7     stav[u]:= CLOSED                        Uzel u je dokončen,
8     i := i + 1; f[u] := i                   dej mu koncovou značku.
```

Nyní určíme časovou složitost algoritmu DFS. Cykly na řádcích 1–3 a 5–7 budou trvat $\Theta(|U|)$, pokud neuvažujeme čas spotřebovaný voláním procedury DFS-Projdi. Tato procedura se volá právě jednou pro každý uzel $v \in U$, neboť se provádí pouze s FRESH uzly a jejím prvním krokem je změnit stav uzlu na OPEN. Během provádění procedury DFS-Projdi(v) se cyklus na řádcích 3–6 provede $|Adj[u]|$-krát. Platí ovšem

$$\sum_{u \in U} |Adj[u]| = \Theta(|H|),$$

takže celková doba provádění řádek 3–6 v proceduře DFS-Projdi je $\Theta(|H|)$. Výsledná složitost celého algoritmu DFS je tedy $\Theta(|U| + |H|)$.

Všimněme si nyní dalších důležitých vlastností algoritmu DFS. Již jsme uvedli, že algoritmem vytvořený p-podles obsahuje obecně několik oddělených stromů. Jeho struktura totiž



Obrázek 4.9: Závorková struktura prohledávání do hloubky

přesně odpovídá hierarchii rekurzivních volání procedury DFS-Projdi: kořenem se stává uzel u, pro který byla procedura vyvolána v hlavní části algoritmu DFS na řádce 7, neboť pro něj se inicializovaná hodnota $p[u] = \text{NIL}$ nemění. Do stromu s tímto kořenem jsou pak zahrnuty všechny uzly objevené při procházení uzlů dostupných z kořene (objevené prostřednictvím seznamů sousedů), což zajišťuje úprava hodnoty $p[v]$ na řádce 5 procedury DFS-Projdi. Platí tedy, že $p[v] = u$ právě tehdy, když byl uzel v objeven v seznamu sousedů uzlu u.

Rekurzivní podoba procedury DFS-Projdi také jasně ukazuje, že časové značky otevření a zavření uzlů vytvářejí **závorkovou strukturu**. Označíme-li okamžik otevření uzlu u jako „$(u$“ a okamžik jeho uzavření jako „$u)$“, potom zápisem průběhu otvírání a zavírání uzlů dostaneme dobře strukturovaný výraz vnořovaných nebo sousedících závorkových struktur. Na obr. 4.9(b) ukazujeme tuto závorkovou strukturu současně s rozsahem jednotlivých podstromů a přiřazenými hodnotami $d[u]$ a $f[u]$ odpovídající prohledání do hloubky grafu z obr. 4.9(a). Jiný způsob vyjádření faktu existence této závorkové struktury podává následující tvrzení.

Věta 4.22: (Závorkový teorém)
Při každém prohledání do hloubky libovolného neorientovaného nebo orientovaného grafu $G = \langle H, U\rangle$ nastává pro libovolnou dvojici uzlů u, v právě jeden z následujících možných případů:

- intervaly $\langle d[u], f[u]\rangle$ a $\langle d[v], f[v]\rangle$ jsou disjunktní
- interval $\langle d[u], f[u]\rangle$ je zcela obsažen v intervalu $\langle d[v], f[v]\rangle$ a uzel u je potomkem uzlu v ve vytvořeném DF-stromu
- interval $\langle d[v], f[v]\rangle$ je zcela obsažen v intervalu $\langle d[u], f[u]\rangle$ a uzel v je potomkem uzlu u ve vytvořeném DF-stromu

Důkaz: Nechť platí $d[u] < d[v]$, pak mohou nastat dva případy podle toho, zda je $d[v] < f[u]$ nebo naopak. V prvním případě je tedy $d[v] < f[u]$, takže uzel v byl otevřen v době, kdy byl uzel u ještě otevřený. To ovšem znamená, že v je potomkem uzlu u, takže všechny z něj vedoucí hrany budou prozkoumány a uzel v bude uzavřen dříve, než se hledání vrátí zpět do uzlu u. V tom případě je ale interval $\langle d[v], f[v]\rangle$ zcela obsažen v intervalu $\langle d[u], f[u]\rangle$.
Ve druhém případě je $f[u] < d[v]$ a z nerovnosti (4.20) plyne, že intervaly $\langle d[u], f[u]\rangle$ a $\langle d[v], f[v]\rangle$ jsou disjunktní.
Pro případ $d[v] < d[u]$ bychom postupovali zcela stejně, pouze bychom zaměnili uzly u a v. △

Důsledek : (Vnořování intervalů potomků)
Uzel v je vlastním potomkem uzlu u v DF-lese (neorientovaného nebo orientovaného) grafu G, právě když platí

$$d[u] < d[v] < f[v] < f[u].$$

Důkaz: Plyne bezprostředně z věty 4.22. △

Následující tvrzení podává další důležitou charakterizaci případu, kdy jeden uzel je potomkem jiného v DF-lese.

Věta 4.23: (Teorém o nové cestě) Uzel v je potomkem uzlu u v DF-lese (neorientovaného resp. orientovaného) grafu $G = \langle H, U \rangle$, právě když v okamžiku $d[u]$ objevení uzlu u je možné sestrojit (neorientovanou resp. orientovanou) cestu z u do v tvořenou výhradně novými (t.j. dosud neobjevenými) uzly.

Důkaz: Pro jediného triviálního potomka, t.j. samotný uzel u se jedná o cestu délky 0, takže se věnujeme jen potomkům vlastním.

$\Rightarrow$: Nechť v je vlastním potomkem uzlu u. Potom pro libovolný uzel x na (jediné) cestě v DF-lese z uzlu u do v podle důsledku 4.22 platí $d[u] < d[x]$. V okamžiku $d[u]$ tedy musí být uzel x ještě neobjevený (t.j. ve stavu FRESH).

$\Leftarrow$: Postupujeme sporem – předpokládejme, že uzel v je dosažitelný z u po cestě tvořené pouze novými uzly, ale přitom se nestane potomkem uzlu u v DF-lese. Bez ztráty na obecnosti můžeme předpokládat, že všechny ostatní uzly na této cestě jsou potomky uzlu u. Nechť w je předchůdce uzlu v na této cestě, takže i w je potomkem uzlu u (jako w si můžeme představit i samotný uzel u). Potom je podle důsledku 4.22 o vnořování $f[w] \leq f[u]$. Jelikož uzel v bude objeven až po objevení uzlu u, ale dříve než je uzavřen uzel w, musí platit

$$d[u] < d[v] < f[w] \leq f[u].$$

Podle věty 4.22 je pak ale nutně interval $\langle d[v], f[v] \rangle$ zcela obsažen v intervalu $\langle d[u], f[u] \rangle$. To ale podle důsledku 4.22 o vnořování znamená, že uzel v je potomkem uzlu u, což je spor. $\triangle$

Další velmi významnou vlastností algoritmu prohledávání do hloubky je možnost využít jej ke klasifikaci hran výchozího grafu $G = \langle H, U \rangle$. Získaná klasifikace hran poskytuje o grafu důležitou informaci – tak např. neexistence tzv. zpětných hran je nutnou a postačující podmínkou acykličnosti grafu (viz lemma 4.26). Prostřednictvím DF-lesa G_p získaného provedením algoritmu prohledávání grafu G do hloubky lze rozlišit následující čtyři typy hran:

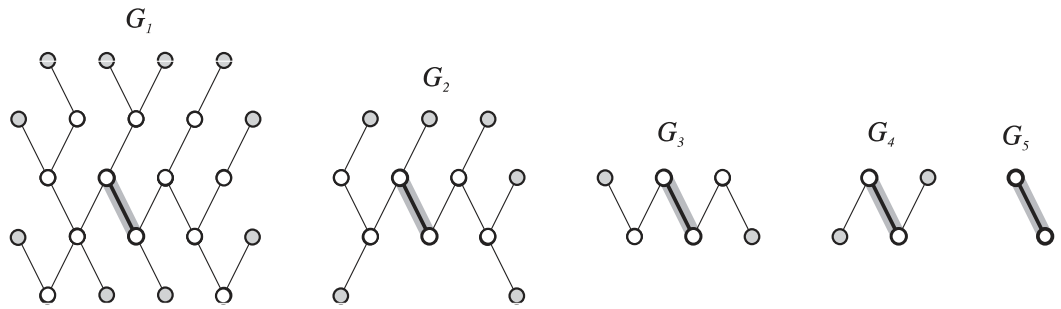
- ◇ **Stromové hrany** jsou hrany tvořící DF-les G_p. Hrana (u, v) je stromovou hranou, jestliže k objevení (otevření) uzlu v došlo při procházení seznamu sousedů uzlu u.
- ◇ **Zpětné hrany** jsou takové nestromové hrany (u, v), které vedou od uzlu u k nějakému jeho předkovi v DF-lese. Případné smyčky považujeme rovněž za zpětné hrany.
- ◇ **Dopředné hrany** jsou takové nestromové hrany (u, v), které vedou od uzlu u k nějakému jeho potomkovi v DF-lese.
- ◇ **Příčné hrany** jsou všechny ostatní hrany. Mohou spojovat dva uzly téhož DF-stromu (pokud jejich počáteční uzel není předkem koncového) nebo uzly ležící v různých DF-stromech.

Na obr. 4.8 a 4.9 je typ každé hrany vyjádřen pomocí označení písmenem **S, Z, D** nebo **P**. Každý graf je možné překreslit tak, aby všechny stromové a dopředné hrany směřovaly v DF-stromu dolů a všechny zpětné hrany nahoru – to vyplývá z možnosti nakreslit v příslušné podobě každý z DF-stromů výsledného lesa.

Určení typu hrany je možné provádět v průběhu hledání, pokud příslušně doplníme algoritmus DFS. Typ hrany (u, v) je totiž možné určit podle stavu uzlu v dosaženého při prvním prozkoumání této hrany (nelze ovšem rozlišit dopředné a příčné hrany):

- FRESH uzel v znamená stromovou hranu
- OPEN uzel v znamená zpětnou hranu
- CLOSED uzel v znamená dopřednou nebo příčnou hranu

Správnost určení typu stromové hrany je zřejmá. Ve druhém případě je třeba si uvědomit, že při prohledávání do hloubky se na začátku procedury DFS-Projdi uzel otevře a pak se procedura volá rekurzivně pro každý nový uzel objevený v jeho seznamu sousedů. Pokud se mezi sousedy uzlu u narazí na již otevřený uzel v, musí to být argument procedury DFS-Projdi v některé z předchozích úrovní jejího rekurzivního volání, a je tedy předkem uzlu u. Poslední případ zřejmě zahrnuje zbývající dvě možnosti typů hran. Rozlišení mezi nimi lze pro hranu



Obrázek 4.10: Prohledávání do hloubky

(u, v) provést podle hodnoty časových značek: je-li $d[u] < d[v]$, jedná se o dopřednou hranu, v případě $d[v] < d[u]$ se jedná o hranu příčnou.

Při procházení neorientovaného grafu a rozlišení typů jednotlivých hran se každá hrana nějak orientuje. Pro neorientovanou hranu $[u, v]$ tak přicházejí v úvahu orientace (u, v) nebo (v, u), použije se ovšem vždy ta orientace, která odpovídá pořadí prvního průchodu touto hranou. Mohlo by se zdát, že takový postup povede k nejednoznačnostem, ale to vylučuje následující tvrzení.

Věta 4.24: Po prohledání do hloubky neorientovaného grafu G bude každá jeho hrana patřit buď mezi stromové nebo mezi zpětné hrany.

Důkaz: Nechť $[u, v]$ je libovolná hrana grafu G, bez ztráty obecnosti můžeme předpokládat, že je $d[u] < d[v]$. Potom musí být uzel v objeven a uzavřen dříve, než bude uzavřen uzel u, neboť je v seznamu sousedů uzlu u. Prochází-li se hrana $[u, v]$ nejdříve ve směru $u \to v$, potom se (u, v) stává stromovou hranou. Prochází-li se hrana $[u, v]$ nejdříve ve směru $v \to u$, potom se stává (v, u) zpětnou hranou, neboť uzel u je ještě otevřený v době, kdy se hrana poprvé prochází. △

Cvičení

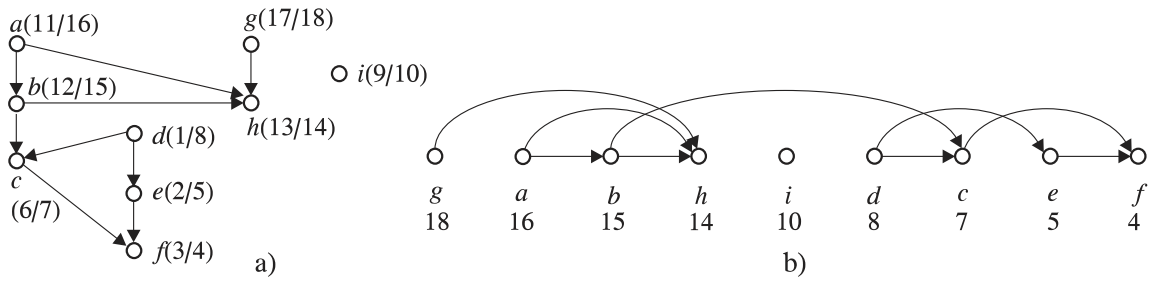
4.3-1. Vyplňte každé políčko (i, j) následující tabulky hodnotou ano/ne podle toho, zda může v některém okamžiku v průběhu prohledávání do hloubky existovat hrana z uzlu ve stavu i do uzlu ve stavu j. Uvažujte zvlášť případ prohledávání neorientovaných a orientovaných grafů.

i / j	FRESH	OPEN	CLOSED
FRESH			
OPEN			
CLOSED			

4.3-2. Simulujte průběh prohledávání do hloubky na grafu z obr. 4.10. Předpokládejte přitom, že uzly se v cyklu na řádcích 5–7 procedury DFS probírají v abecedním pořadí a že v abecedním pořadí jsou rovněž uzly uspořádány v seznamech sousedů ve spojové reprezentaci. Určete také časové značky otevření a uzavření všech uzlů a typ (S, Z, D, P) každé hrany.

4.3-3. Dokažte, hrana (u, v) je
(a) stromovou hranou, právě když $d[u] < d[v] < f[v] < f[u]$
(b) zpětnou hranou, právě když $d[v] < d[u] < f[u] < f[v]$
(c) příčnou hranou, právě když $d[v] < f[v] < d[u] < f[u]$

4.3-4. Nalezněte protipříklad, který vyvrátí následující tvrzení: Pokud v orientovaném grafu G existuje orientovaná cesta z uzlu u do uzlu v a platí $d[u] < d[v]$ po prohledání grafu G do hloubky, potom je uzel v potomkem uzlu u na vytvořeném DF-lese.



Obrázek 4.11: Topologické uspořádání

4.3-5. Upravte algoritmus prohledávání do hloubky tak, aby se během jeho provádění vypisovaly všechny hrany orientovaného grafu G spolu s označením jejich typu. Uveďte, jak se musí takto upravený algoritmus (případně) změnit, aby byl použitelný pro neorientovaný graf.

4.3-6. Vysvětlete, jak je možné, aby uzel u orientovaného grafu skončil po prohledání do hloubky jako jediný uzel nějakého DF-stromu, přestože do u vcházejí i z něj vystupují hrany.

4.3-7. Algoritmus prohledání neorientovaného grafu G do hloubky lze použít k určení komponent grafu G, neboť DF-les obsahuje právě tolik DF-stromů, kolik má G komponent. Upravte algoritmus DFS tak, aby pro každý uzel u určil hodnotu $k[u]$ z intervalu $\langle 1, p\rangle$, kde p je počet komponent grafu G, přičemž $k[u] = k[v]$, právě když uzly u a v leží ve stejné komponentě.

4.4 Topologické uspořádání

Algoritmus prohledávání do hloubky poskytuje jako vedlejší produkt údaje, které lze využít při řešení mnoha grafových úloh – ukážeme to na problému **topologického uspořádání** uzlů orientovaného grafu $G = \langle H, U\rangle$. Topologickým uspořádáním se rozumí takové úplné uspořádání uzlů, v němž je pro každou hranu (u, v) uzel u před uzlem v. Takové uspořádání lze zřejmě zavést pouze pro acyklické grafy (viz odst. 2.2), neboť libovolný cyklus (včetně smyčky) by takové uspořádání znemožnil. Topologické uspořádání grafu můžeme vyjádřit tak, že uzly seřadíme do jediné horizontální úrovně, a orientace všech hran pak musí směřovat jedním směrem – např. zleva doprava. Na obr. 4.11b ukazujeme v této podobě výsledek topologického uspořádání grafu z obr. 4.11a (za označením uzlu je vždy uvedena dvojice $(d[u], f[u])$). Jednoduchý algoritmus topologického uspořádání acyklického orientovaného grafu lze s použitím procedury DFS formulovat takto:

Algoritmus 4.25 Topologické uspořádání uzlů grafu

TOP-SORT-1(G)

1 vytvoř prázdný seznam uzlů S
2 pomocí DFS(G) počítej okamžiky uzavření $f[v]$ všech uzlů grafu G
3 v okamžiku uzavírání ulož každý uzel v na začátek seznamu uzlů S
4 seznam S obsahuje uzly seřazené podle požadavků topologického uspořádání

Na obr. 4.11b je vidět, že topologické uspořádání uzlů odpovídá jejich sestupnému seřazení podle okamžiku uzavření $f[v]$ v algoritmu prohledání do hloubky. Není třeba připomínat, že topologické uspořádání grafu není obecně určeno jednoznačně.

Časová složitost popsaného postupu je zřejmě $O(|U| + |H|)$, neboť prohledání do hloubky trvá $O(|U| + |H|)$ a k přidání nového uzlu na začátek seznamu potřebujeme $O(1)$ pro každý z $|U|$ uzlů zpracovávaného grafu. Správnost uvedeného postupu dokážeme pomocí následujícího tvrzení, které charakterizuje orientované acyklické grafy.

Lemma 4.26: Orientovaný graf G je acyklický právě tehdy, když se při jeho prohledání do hloubky neobjeví žádná zpětná hrana.

Důkaz: $\Rightarrow$: Předpokládejme existenci nějaké zpětné hrany (u, v). Pak je ale uzel v předkem uzlu u na DF-lese, a tedy existuje orientovaná cesta z uzlu v do uzlu u. Složením této cesty a hrany (u, v) by však vznikl cyklus, což je spor.

$\Leftarrow$: Nechť G obsahuje nějaký cyklus C, ukážeme, že při prohledání do hloubky se pak nalezne nějaká zpětná hrana. Označme jako v první uzel cyklu C, který bude při prohledání objeven, a nechť (u, v) je hrana cyklu C vedoucí do uzlu v. V okamžiku $d[v]$ existuje cesta z uzlu v do uzlu u tvořená pouze novými uzly, takže podle věty 4.23 je u potomkem uzlu v na DF-lese. Hrana (u, v) je tedy zpětnou hranou. △

Věta 4.27: Procedurou TOP-SORT-1(G) se získá topologické uspořádání uzlů orientovaného acyklického grafu G.

Důkaz: Předpokládejme, že se pomocí algoritmu DFS na daném grafu $G = \langle H, U\rangle$ získají časové značky uzavření uzlů. Stačí dokázat, že pokud pro libovolnou dvojici uzlů $u, v \in U$ existuje hrana $(u, v) \in H$, potom je $f[v] < f[u]$. Uvažujme libovolnou hranu (u, v) prozkoumávanou v průběhu provádění algoritmu DFS(G). Uzel v nemůže být v tom okamžiku ještě otevřený, neboť pak by byl v předkem uzlu u a (u, v) by byla zpětná hrana, což by bylo ve sporu s lemmatem 4.26. Uzel v tedy musí být buď nový nebo uzavřený. Je-li nový, stane se potomkem uzlu u a platí $f[v] < f[u]$. Je-li v uzavřený, platí opět $f[v] < f[u]$. Pro libovolnou hranu (u, v) grafu G tedy máme $f[u] < f[v]$, což dokazuje naše tvrzení. △

Pro testování acykličnosti a současně pro topologické uspořádání grafu se nabízí ještě další postupy. Zcela neefektivní by ovšem bylo např. systematicky procházet všechny orientované cesty grafu a testovat, zda některá z nich nelze uzavřít, a vytvořit tak cyklus. Rozumnější je hledat v grafu G podgraf splňující nějakou jednoduše ověřitelnou podmínku pro existenci cyklu. Takovou snadno testovatelnou podmínku nabízí následující tvrzení.

Věta 4.28: Nechť pro uzly neprázdného orientovaného grafu $G = \langle H, U\rangle$ platí

$$\delta^+_G(u) \geq 1 \quad \text{pro všechna } u \ \in U \tag{4.21}$$

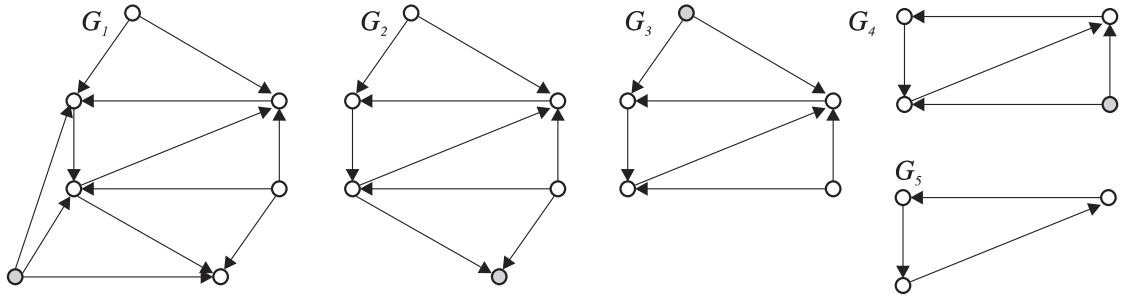
(tzn. graf G nemá žádný list). Potom graf G obsahuje cyklus.

Důkaz: Zvolme libovolný uzel u_0 grafu G. Z podmínky (4.21) plyne, že z uzlu u_0 vystupuje alespoň jedna hrana, označme u_1 její koncový uzel. Stejný argument uplatníme i pro uzel u_1 a dostaneme se do uzlu u_2, atd. Vytváříme posloupnost uzlů $\langle u_0, u_1, u_2, \ldots\rangle$ tak dlouho, až se v ní nějaký uzel objeví podruhé (to musí vzhledem ke konečnosti grafu G někdy nastat). Tím jsme však nalezli požadovaný cyklus. △

Při přechodu k opačně orientovanému grafu se podmínka (4.21) bude týkat vstupních stupňů $\delta^-_G(u)$, takže tvrzení věty 4.28 je možné rozšířit: neprázdný orientovaný graf obsahuje cyklus, jestliže nemá žádný list nebo žádný kořen. Pro existenci cyklu v grafu navíc stačí, aby uvedenou podmínku splňoval nějaký jeho podgraf – ukážeme teď velmi jednoduchý postup, který takový podgraf určí nebo jeho existenci vyloučí. Základem postupu je následující tvrzení, jehož důkaz ponecháme jako cvičení.

Věta 4.29: Orientovaný graf G je acyklický právě tehdy, je-li acyklický každý jeho podgraf $G - \{u\}$, kde u je libovolný kořen nebo list grafu G.

Při zjišťování acykličnosti tedy stačí vypustit z grafu nějaký jeho kořen nebo list, potom jiný kořen nebo list vzniklého podgrafu atd., až dospějeme k prázdnému grafu nebo k podgrafu, který už žádný kořen ani list nemá. Tento postup ilustrujeme na postupné redukci grafu na obr. 4.12 – jelikož poslední získaný podgraf je neprázdný, výchozí graf není acyklický. V souladu s rozšířeným zněním věty 4.28 ovšem stačí vypouštět např. pouze kořeny grafu. To je základní myšlenka následujícího algoritmu, který provede topologické uspořádání uzlů grafu G, pokud je acyklický, jinak ohlásí neúspěch.



Obrázek 4.12: Testování acykličnosti grafu

V algoritmu se používá pomocné pole $\delta[u]$, které obsahuje vstupní stupeň uzlů v podgrafech získávaných postupným vypouštěním kořenů. Kromě toho se kořeny ukládají do množiny M, z níž se vybírají pro úpravu stupňů následníků, a do zásobníku, v němž se získá po ukončení topologicky uspořádaná posloupnost uzlů.

Algoritmus 4.30 Topologické uspořádání uzlů grafu

TOP-SORT-2(G)

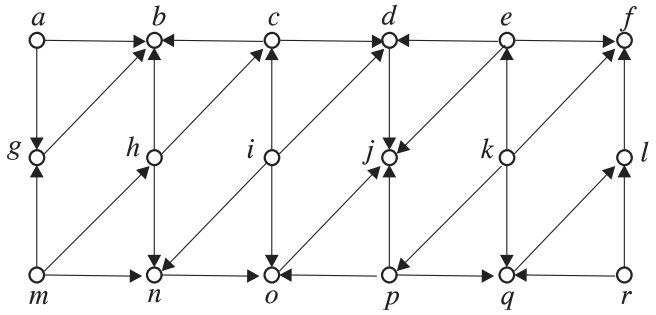
```
1   for každý uzel u ∈ U                    Počáteční nastavení
2       do δ[u] := 0                        δ(u) všech uzlů na nulu.
3   for každý uzel u ∈ U do                 Výpočet skutečné
4       for každý uzel v ∈ Adj[u] do        hodnoty δ[v]
5           δ[v] := δ[v] + 1                pro každý uzel.
6   M := ∅; INIT_STACK                      Vyprázdni M a zásobník.
7   for každý uzel u ∈ U do                 Zjisti kořeny
8       if δ[u] = 0                         a ulož je
9           then  M := M ∪ {u}              do množiny M
10                PUSH(u)                   i do zásobníku.
11  while M ≠ ∅ do                          Dokud je co,
12      v := libovolný uzel z M             vybírej z množiny M
13      M := M − {v}                        a pro vybraný prvek
13      for každý uzel w ∈ Adj[v] do        uprav δ[w] jeho následníků.
14          δ[w] := δ[w] − 1
15          if δ[w] = 0                     Nově vzniklý kořen
16              then M := M ∪ {w}           se přidá do M
17                   PUSH(w)                i do zásobníku.
18  if zásobník neobsahuje všechny uzly
19      then topologické uspořádání neexistuje
```

Analýza časové složitosti tohoto algoritmu je opět jednoduchá. Inicializační cykly na řádcích 1–2, resp. 3–5 trvají $O(|U|)$, resp. $O(|H|)$, vyhledání prvních kořenů na řádcích 7–10 trvá opět $O(|U|)$. Jelikož se každý uzel může dostat do množiny M nejvýše jednou, proběhne hlavní cyklus začínající na řádce 11 nejvýše $|U|$-krát, přičemž vnořený cyklus úpravy stupňů proběhne pro uzel w právě $|Adj[w]|$-krát. Ukládání do množiny i do zásobníku se provede v čase $O(1)$, takže celková složitost hlavního cyklu je $O(|H|)$. Sečtením uvedených hodnot dostáváme tedy stejnou celkovou složitost jako pro algoritmus využívající prohledávání do hloubky – tedy $O(|U|+|H|)$.

Cvičení

4.4-1. Ukažte výsledek topologického uspořádání grafu na obr. 4.13 s použitím obou uvedených algoritmů. Procházení uzlů a seznamů sousedů provádějte opět v abecedním pořadí.

4.4-2. Pro zadaný acyklický graf existuje zpravidla více než jedno topologické uspořádání jeho uzlů, některá z nich dokonce nelze získat provedením algoritmu TOP-SORT-1. Ukažte příklad



Obrázek 4.13: Topologické uspořádání orientovaného grafu

acyklického grafu G a jeho topologického uspořádání, které nemůže být výsledkem provedení tohoto algoritmu, ať je pořadí uzlů v seznamech sousedů jakékoliv. Ukažte jiný příklad grafu, ve kterém přes různá pořadí uzlů v seznamech sousedů vede tento algoritmus na stejné výsledné uspořádání uzlů.

4.4-3. Navrhněte obecný postup orientace hran neorientovaného grafu, jehož výsledkem je zaručeně acyklický graf.

4.4-4. Navrhněte algoritmus, který určí, zda zadaný neorientovaný graf $G = \langle H, U \rangle$ obsahuje kružnici. Časová složitost tohoto algoritmu by měla být nezávislá na počtu hran, tedy $O(|U|)$.

4.4-5. Potvrďte nebo vyvraťte následující tvzení: Pokud obsahuje orientovaný graf G cykly, pak výsledek algoritmu TOP-SORT-1(G) poskytne takové uspořádání uzlů grafu, které minimalizuje počet hran, jejichž orientace neodpovídá požadavku topologického uspořádání.

4.5 Silné komponenty

Již jsme upozornili na to, že prohledáním do hloubky lze určit strukturu komponent neorientovaného grafu, tedy i jeho souvislost. Pro orientované grafy má obdobnou důležitost určení jejich silných komponent. Mnoho grafových úloh se totiž řeší rozkladem na podproblémy odpovídající jednotlivým silným komponentám s následnou kombinací výsledků těchto podproblémů. Takový postup naznačilo i zavedení pojmu kondenzace orientovaného grafu v odstavci 2.2.

Je možné očekávat, že prohledání do hloubky lze využít i pro určení struktury silných komponent. Připomeňme nejprve, že silná komponenta je každý maximální silně souvislý podgraf daného grafu – tedy podgraf, v němž jsou každé dva uzly propojitelné v obou směrech orientovanou cestou. Návrh následujícího algoritmu se opírá o důležitou vlastnost: orientovaný graf $G = \langle H, U \rangle$ i k němu opačně orientovaný graf $G^- = \langle H^{-1}, U \rangle$ mají (pochopitelně až na orientaci hran) stejnou strukturu svých silných komponent.

Vytvoření opačně orientovaného grafu ke grafu $G = \langle H, U \rangle$ má složitost $O(|U| + |H|)$. Stejnou asymptotickou složitost má i následující algoritmus určení silných komponent, který sestává ze dvou prohledání do hloubky – nejprve se prochází graf G a potom opačně orientovaný graf G^-.

Algoritmus 4.31 Rozklad na silné komponenty

S-COMP(G)

1 proveď DFS(G), který určí časovou značku uzavření $f[u]$ každého uzlu u
2 vytvoř opačně orientovaný graf G^-
3 proveď DFS(G^-), přitom v hlavním cyklu DFS probírej uzly v pořadí klesající hodnoty $f[u]$ (získané v kroku 1)
4 rozklad množiny uzlů podle jednotlivých stromů DF-lesa získaného v kroku 3 určuje rozklad na silné komponenty

Z formulace algoritmu není na první pohled zřejmé, proč se struktura silných komponent získá právě takto. Abychom to mohli dokázat, odvodíme některé další vlastnosti prohledávání do hloubky.

Věta 4.32: Po prohledání orientovaného grafu do hloubky budou všechny uzly téže silné komponenty ležet ve stejném DF-stromu.

Důkaz: Uvažujme libovolnou silnou komponentu grafu a označme jako r uzel, který bude při prohledání do hloubky otevřen v této silné komponentě jako první. To znamená, že v okamžiku jeho otevření budou všechny ostatní uzly této komponenty ještě neobjevené. Všechny cesty spojující uzel r s ostatními uzly silné komponenty jsou tedy tvořeny novými uzly. Podle věty 4.23 budou všechny tyto uzly potomky uzlu r na DF-stromu vytvořeném při prohledání. △

Ve zbytku tohoto odstavce budeme pomocí $d[u]$ a $f[u]$ označovat časové okamžiky otevření a zavření uzlu u při provádění řádky 1 algoritmu S-COMP, tedy při procházení do hloubky grafu G. Podobně i symbol $u \longrightarrow v$ bude vyjadřovat dostupnost uzlu v z uzlu u v grafu G, nikoliv v grafu G^-.

K důkazu správnosti algoritmu S-COMP zavedeme pojem **praotce** $\pi(u)$ uzlu u: je to takový uzel w, který je ze všech uzlů dostupných z u uzavřen nejpozději při prohledávání na řádce 1. Je tedy možné psát

$$\pi(u) \;=\; w \;:\; (u \longrightarrow w) \;\&\; f[w] \text{ je maximální}$$

Tato definice nevylučuje možnost $\pi(u) = u$, neboť také uzel u je dostupný z u (cestou nulové délky), takže platí

$$f[u] \leq f[\pi(u)] \tag{4.22}$$

Každý praotec nějakého uzlu je současně svým vlastním praotcem, neboli $\pi(\pi(u)) = \pi(u)$. Pro libovolné dva uzly $u, v \in U$ totiž platí implikace

$$u \longrightarrow v \quad \Rightarrow \quad f[\pi(v)] \leq f[\pi(u)], \tag{4.23}$$

neboť množina $\{w : v \longrightarrow w\}$ je podmnožinou množiny $\{w : u \longrightarrow w\}$ a praotec má nejpozdější okamžik uzavření ze všech dostupných uzlů. Jelikož podle definice praotce je $u \longrightarrow \pi(u)$, dostáváme podle vztahu (4.23) nerovnost $f[\pi(\pi(u))] \leq f[\pi(u)]$. Současně je ale $f[\pi(u)] \leq f[\pi(\pi(u))]$ podle vztahu (4.22). Platí tedy $f[\pi(\pi(u))] = f[\pi(u)]$, ale to znamená $\pi(\pi(u)) = \pi(u)$, neboť přiřazení časových značek uzavírání uzlů je prosté zobrazení.

Důležitost praotce spočívá v tom, že pro nás bude reprezentantem celé silné komponenty. Každá silná komponenta obsahuje totiž uzel, který je praotcem všech jejích uzlů. Který konkrétní uzel se stane praotcem, záleží na průběhu prohledávání, je to ale vždy první otevřený uzel silné komponenty, který bude zároveň jejím posledním uzavřeným uzlem. Při prohledávání opačně orientovaného grafu G^- pak bude tento uzel vybrán jako kořen DF-stromu obsahujícího všechny uzly této silné komponenty. Uvedené vlastnosti nyní dokážeme.

Věta 4.33: Praotec $\pi(u)$ libovolného uzlu $u \in U$ při libovolném prohledávání grafu $G = \langle H, U \rangle$ do hloubky je předkem uzlu u.

Důkaz: Pro $\pi(u) = u$ tvrzení platí, předpokládejme tedy $\pi(u) \neq u$. Uvažujme o tom, v jakém stavu se může nacházet praotec $\pi(u)$ v okamžiku otevření $d[u]$ uzlu u. Nemůže být uzavřený, neboť pak by bylo $f[\pi(u)] < f[u]$, což odporuje nerovnosti (4.22). Je-li otevřený, pak je předkem uzlu u a tvrzení platí.

Zbývá tedy dokázat, že $\pi(u)$ není v té době ještě neobjeveným uzlem. Můžeme rozlišit dva případy podle stavu uzlů na cestě z u do $\pi(u)$.

(i) Jsou-li všechny uzly na této cestě dosud nové (t.j. neobjevené), pak se podle věty o nové cestě stane uzel $\pi(u)$ potomkem uzlu u. To ale znamená, že $f[\pi(u)] < f[u]$, což je ve sporu se vztahem (4.22).

(ii) Jsou-li některé z vnitřních uzlů cesty již objeveny, označme jako t ten z nich, který je nejblíže

k $\pi(u)$. Pak nesmí být t uzavřený, neboť jeho následníkem je nový uzel a z uzavřeného uzlu nikdy nemůže vést hrana k novému uzlu. Uzel t je tedy otevřený a existuje z něj nová cesta do uzlu $\pi(u)$, takže podle věty o nové cestě bude $\pi(u)$ potomkem uzlu t. Bude tedy $f[t] > f[\pi(u)]$, což je ale ve sporu s výběrem praotce $\pi(u)$, neboť t je dostupný z u. △

Důsledek 1: Při libovolném prohledání orientovaného grafu $G = \langle H, U\rangle$ se pro každý uzel $u \in U$ nachází jeho praotec $\pi(u)$ ve stejné silné komponentě jako u.

Důkaz: Podle definice praotce je $u \longrightarrow \pi(u)$ a z přechozí věty plyne $\pi(u) \longrightarrow u$, neboť $\pi(u)$ je předkem uzlu u. △

Následující věta potvrzuje oprávněnost našeho výběru praotců jako reprezentantů silných komponent.

Věta 4.34: Uzly $u, v \in U$ orientovaného grafu $G = \langle H, U\rangle$ se nacházejí ve stejné silné komponentě grafu G právě tehdy, mají-li stejného praotce při prohledání grafu G do hloubky.

Důkaz: $\Rightarrow$: Předpokládejme, že u a v jsou ve stejné silné komponentě. Protože je $u \longrightarrow v$ a současně i $v \longrightarrow u$, jsou množiny uzlů dostupných z uzlu u i z uzlu v stejné, takže mají i stejného praotce.

$\Leftarrow$: Nechť je $\pi(u) = \pi(v)$. Podle důsledku věty 4.33 je $\pi(u)$ ve stejné silné komponentě jako u a $\pi(v)$ je ve stejné silné komponentě jako v. Jsou tedy u i v ve stejné silné komponentě. △

Nyní je již struktura algoritmu S-COMP srozumitelnější. Každou silnou komponentu tvoří množina uzlů se stejným praotcem. Podle věty 4.33 a závorkové věty 4.22 bude praotec při prohledání na řádce 1 algoritmu S-COMP jak prvním otevřeným uzlem, tak současně i posledním uzavřeným uzlem své silné komponenty.

Zbývá osvětlit, proč se na řádce 3 algoritmu S-COMP prochází do hloubky graf G^-. Označme jako r poslední uzel uzavřený během procházení do hloubky na řádce 1. Tento uzel je praotcem – je sám ze sebe dostupný a žádný jiný uzel nemá pozdější okamžik uzavření. Do silné komponenty určené tímto praotcem pak budou patřit všechny uzly, které jej mají za praotce, tedy uzly, ze kterých je r dostupný, ale není z nich dostupný žádný uzel s vyšší hodnotou časové značky uzavření nežli $f[r]$. Hodnota $f[r]$ je však nejvyšší pro celý graf G, takže to jsou prostě jen uzly, z nichž je r dostupný. To je ale právě množina uzlů, které jsou dostupné z r v opačně orientovaném grafu G^-! Prohledání do hloubky na řádce 3 tedy začne objevením a uzavřením všech uzlů náležících do silné komponenty s praotcem r. Je současně vidět, že tento úkol by mohlo splnit i prohledání do šířky nebo jiný algoritmus prohledání, který nalezne dostupné uzly.

Po určení všech uzlů silné komponenty s praotcem r bude hledání na řádce 3 pokračovat otevřením uzlu r' s nejvyšší hodnotou $f[r']$ po odebrání všech uzlů předchozí silné komponenty. Také uzel r' je svým vlastním praotcem, neboť z něj není dostupný žádný uzel s pozdějším okamžikem uzavření (jinak by byl zahrnut do silné komponenty praotce r). Ze stejných důvodů jako dříve je zřejmé, že každý dosud neuzavřený uzel, z něhož je r' dostupný, musí patřit do silné komponenty s praotcem r'. Prohledávání grafu G^- na řádce 3 vycházející z r' tedy nalezne a uzavře všechny uzly silné komponenty s praotcem r'.

Tímto způsobem se při hledání do hloubky na řádce 3 uzavírá jedna silná komponenta po druhé. Praotcem každé z nich je uzel, který je argumentem volání procedury DFS-Projdi na řádce 7 – je to její první otevřený uzel. Rekurzivní volání procedury DFS-Projdi pak uzavřou všechny uzly této silné komponenty a na nejvyšší úrovni se pak nakonec uzavře i samotný praotec. Hlavní cyklus na řádce 5 v DFS pak nalezne dosud neuzavřený uzel s nejvyšší hodnotou časové značky uzavření, se kterým pak pokračuje určení další silné komponenty. Po tomto rozboru přistoupíme k závěrečnému tvrzení.

Věta 4.35: Provedením algoritmu S-COMP(G) se určí struktura silných komponent orientovaného grafu G.

Důkaz: Postupujeme indukcí podle pořadí DF-stromu nalezeného při prohledávání grafu G^- do hloubky – tvrdíme, že pokud všechny předchozí získané DF-stromy určovaly silné komponenty, pak i uzly dalšího DF-stromu určují silnou komponentu.
1. Pro první DF-strom žádné předchozí stromy nemáme, a přitom víme (viz předchozí rozbor), že určuje silnou komponentu. Indukční hypotéza je tedy triviálně splněna.

2. Uvažujme DF-strom T s kořenem r vytvořený při prohledání grafu G^-. Nechť $P(r)$ označuje všechny uzly s praotcem r, tedy

$$P(r) = \{v \in U : \pi(v) = r\}.$$

Dokážeme, že do T se dostanou právě a pouze uzly z $P(r)$.
$\Leftarrow$: Podle věty 4.32 se všechny uzly z $P(r)$ dostanou do jednoho DF-stromu. Protože $r \in P(r)$ a r je kořen stromu T, všechny uzly z $P(r)$ budou v T.
$\Rightarrow$: Ukážeme, že do T se nemůže dostat žádný uzel w, pro který je $f[\pi(w)] > f[r]$ nebo $f[\pi(w)] < f[r]$. Podle indukčního předpokladu se uzel w, pro který je $f[\pi(w)] > f[r]$, musel dostat do DF-stromu s kořenem $\pi(w)$, který byl nalezen dříve, takže se nemůže dostat do stromu T. Ani uzel, pro který je $f[\pi(w)] < f[r]$, se nemůže dostat do T. To by totiž znamenalo, že $w \longrightarrow r$ a podle vztahu (4.23) s použitím $r = \pi(r)$ bychom dostali $f[\pi(w)] \geq f[\pi(r)] = f[r]$, což je ve sporu s předpokladem $f[\pi(w)] < f[r]$.

Strom T tedy obsahuje právě ty uzly, které mají praotce r, takže určuje tutéž silnou komponentu jako $P(r)$. △

Cvičení

4.5-1. Jak se může změnit počet silných komponent orientovaného grafu, přidáme-li jednu hranu?

4.5-2. Simulujte provedení algoritmu S-COMP na grafu z obr. 4.10. Vyznačte přitom především časové značky uzavírání uzlů $f[u]$ získané v průběhu prvního průchodu grafem na řádce 1, a dále výsledný DF-les pořízený během druhého prohledání na řádce 3. Předpokládejte, že uzly se probírají v abecedním pořadí, v němž jsou také uspořádany seznamy sousedů.

4.5-3. Uvažujte zjednodušení algoritmu S-COMP, při němž se druhé prohledání provede opět v původním grafu (ne v opačně orientovaném), ale v pořadí rostoucích hodnot $f[u]$. Je toto zjednodušení přípustné?

4.5-4. Navrhněte algoritmus pro kondenzaci orientovaného grafu $G = \langle H, U\rangle$, který pracuje v čase $O(|H|+|U|)$. Algoritmus musí vyloučit existenci rovnoběžných hran ve výsledném grafu.

4.5-5. Popište způsob, jak pro zadaný orientovaný graf $G = \langle H, U\rangle$ vytvořit faktor $G' = \langle H', U\rangle$, který splňuje všechny následující podmínky:
(a) má stejnou strukturu silných komponent jako graf G
(b) má stejnou kondenzaci jako graf G
(c) má minimální počet hran.

4.5-6. Orientovaný graf $G = \langle H, U\rangle$ se nazývá **polosouvislý**, pokud pro každou dvojici jeho uzlů $u, v \in U$ platí buď $u \longrightarrow v$ nebo $v \longrightarrow u$. Navrhněte efektivní algoritmus pro testování polosouvislosti grafu, dokažte jeho správnost a odvoďte časovou složitost.

4.5-7. Klasifikace hran při prohledání do šířky
Podobně jako při prohledávání do hloubky je i při prohledávání do šířky možné rozčlenit hrany dostupné z výchozího uzlu do čtyř skupin: stromové, zpětné, dopředné a příčné.

(a) Dokažte, že při prohledání neorientovaného grafu do šířky platí následující tvrzení:

1. Žádné hrany nebudou označeny jako zpětné nebo dopředné.
2. Pro každou stromovou hranu (u, v) platí $d[v] = d[u] + 1$.
3. Pro každou příčnou hranu (u, v) platí $d[v] = d[u]$ nebo $d[v] = d[u] + 1$.

(b) Dokažte, že při prohledání orientovaného grafu do šířky platí následující tvrzení:

1. Žádné hrany nebudou označeny jako dopředné.
2. Pro každou stromovou hranu (u, v) platí $d[v] = d[u] + 1$.
3. Pro každou příčnou hranu (u, v) platí $d[v] \leq d[u] + 1$.
4. Pro každou zpětnou hranu (u, v) platí $0 \leq d[v] < d[u]$.