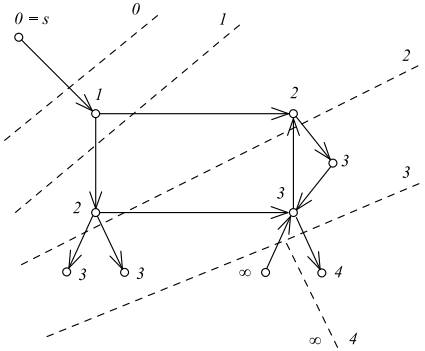
3. Prohledávání grafů

Prohledání do šířky *Breadth-First Search – BFS*

Jde o grafový algoritmus, který postupně prochází všechny vrcholy v dané komponentě souvislosti. Algoritmus nejprve projde všechny sousedy počátečního vrcholu, poté sousedy sousedů, atd... Díky tomuto způsobu procházení se někdy též nazývá „*algoritmus vlny*“, neboť se z počátečního vrcholu šíří pomyslná vlna, která v každém kroku nalezne všechny uzly, které mají od počátečního vrcholu stejnou vzdálenost. Algoritmus se tedy skvěle hodí například pro hledání nejkratší cesty mezi dvěma vrcholy v grafu.

Zatím předpokládejme, že graf, se kterým pracujeme, je orientovaný. Orientovanou hranu (u, v) z u do v budeme obvykle zkracovat jako uv. Pro neorientované grafy bude vše obdobné.



Prasečí graf a průchod vlny skrz něj

Popis algoritmu: Na začátku vložíme do fronty Q počáteční vrchol v_0. Dále si v poli Z budeme pro každý vrchol pamatovat značku, zda jsme ho již navštívili ($Z[v] = 1$), či nikoli ($Z[v] = 0$). Na počátku jsou všechny značky nulové, jen vrchol v_0, který je označen a vložen do fronty.

V každém dalším kroku pak zkoumáme frontu Q: pokud není prázdná, vezmeme z ní první vrchol u a podíváme se na všechny vrcholy v, do nichž z u vede hrana. Pokud sousedi ještě nejsou označení, tak je označíme a přidáme je do fronty k následnému zpracování. Toto opakujeme, dokud není fronta prázdná.

Algoritmus:

1. $Q \leftarrow \{v_0\}$.
2. $Z[*] \leftarrow 0, Z[v_0] \leftarrow 1$.
3. Dokud $Q \neq \emptyset$ opakujeme:
4. Vyzvedneme vrchol u z Q.
5. $\forall v : uv \in E$:
6. Je-li $Z[v] = 0 \Rightarrow Z[v] \leftarrow 1$, přidáme v do Q.

Pozorování: *BFS* se zastaví.

Důkaz: Zpracováváme jen vrcholy, které byly ve frontě. Každý vrchol se dostane do fronty maximálně jednou. (Každý je označen max. jednou, značky neodstraňujeme.)

Lemma: BFS(v_0) označí v právě tehdy, když existuje cesta z v_0 do v.

Důkaz: „$\Longrightarrow$“: Platí jako invariant po celou dobu běhu algoritmu. To dokážeme indukcí dle doby běhu algoritmu:

První krok indukce je triviální, neboť cesta z v_0 do v_0 existuje vždy. Nyní si představme, že označujeme vrchol v přes hranu uv. To znamená, že vrchol u již musel být označený. Dle indukčního předpokladu tedy existuje cesta z v_0 do u, a tudíž pokud k této cestě „přilepíme“ hranu uv, dostáváme sled z v_0 do v, který lze vždy zredukovat na cestu.

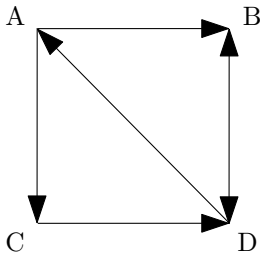
„$\Longleftarrow$“ Sporem: Nechť existuje neoznačený vrchol v dosažitelný po nějaké cestě z v_0. Uvažme nejkratší cestu (v_0, v): $v_0, v_1 \ldots, v_k = v$ a vezměme minimální i takové, že v_i není označený. Víme, že $i > 0$ (neboť v_0 je určitě označen už na začátku algoritmu). Tudíž v_{i-1} je označený. Při označení jsme ho ovšem přidali do fronty, takže jsme ho z fronty museli později zase vyjmout. Při tom jsme ovšem museli objevit hranu $v_{i-1}v_i$ a označit vrchol v_i, což je spor. ♡

Nyní tedy víme, že algoritmus je správný, a máme představu o tom, jak funguje. Podíváme-li se na něj podrobněji, zjistíme, že je hodně závislý na tom, jak si budeme graf pamatovat. Zanedlouho zároveň zjistíme, že nám reprezentace grafu v paměti znatelně ovlivní časovou (i paměťovou) složitost celého algoritmu.

Reprezentace grafu v paměti

Mějme nějaký orientovaný graf G s n vrcholy a m hranami. Jak ho reprezentovat?

Vrcholy můžeme očíslovat od 1 do n. Pro uložení hran máme na výběr hned několik způsobů. Předvedeme si je na grafu z následujícího obrázku:



Ukázkový graf

1. matice sousednosti

Matice sousednosti pro graf G na n vrcholech je čtvercová matice A o rozměrech $n \times n$, taková, že $A_{i,j}$ popisuje, jestli z vrcholu i do vrcholu j vede hrana ($A_{i,j} = 1$) nebo nikoliv ($A_{i,j} = 0$).

Náš graf z obrázku výše by tedy v maticové reprezentaci vypadal takto:

$$\begin{array}{c} \\ A \\ B \\ C \\ D \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

S touto maticí se pracuje velmi snadno, např. všechny sousedy i-tého vrcholu zjistíme jednoduše tak, že projdeme i-tý řádek matice a najdeme všechny jedničky. Má ovšem dvě zřejmé nevýhody: časovou a paměťovou složitost. Projití sousedů jednoho vrcholu trvá vždy $\Theta(n)$, projití sousedů pro všechny vrcholy (což potřebujeme v BFS) pak trvá $\Theta(n^2)$. Velikost matice je vždy $n \times n$, bez ohledu na to, jak „řídký" je graf. U grafu s mnoha vrcholy, ale s malým počtem hran, tedy budeme zbytečně plýtvat místem v paměti. Tato reprezentace je tedy nevýhodná především pro třídy grafů, jako jsou stromy, které mají $n-1$ hran, nebo rovinné grafy, které mají nejvýše $3n-6$ hran.

Pozorování: BFS s reprezentací maticí sousednosti běží v čase: $\Theta(n^2)$.

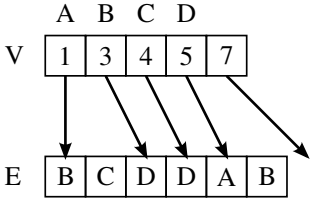
Důkaz: Už jsme si uvědomili, že každý vrchol se dostane do fronty Q nejvýše jednou. Pro každý vrchol ve frontě potřebujeme projít jeho sousedy, což nám trvá s reprezentací maticí sousednosti $\Theta(n)$. Vrcholů je celkem n, tedy časová složitost je $\Theta(n^2)$. ♡

2. seznam sousedů

V matici sousednosti jsme museli procházet jak hrany, tak nehrany, což bylo zbytečné. Bylo by tedy výhodnější pamatovat si pro každý vrchol pouze jeho sousedy. To můžeme zařídit například jedním ze dvou následujících způsobů:

Uchovávejme pole indexované vrcholy tak, že v každém prvku pole je ukazatel na spojový seznam sousedů tohoto vrcholu. Tedy $L(v) = \{w : vw \in E(G)\}$.

Pokud se nám nebude chtít pracovat se spojovými seznamy, můžeme využít reprezentaci pomocí dvou polí. První pole V bude opět indexované vrcholy. V druhém poli E budou pro každý vrchol uloženi jeho sousedé. V poli V si pamatujeme pro každý vrchol i index do pole E, kde začínají jeho sousedé, a navíc dodefinujeme, že $V[n+1] = m+1$. K sousedům vrcholu i se pak již dostaneme snadno – nalezneme je na pozicích $V[i], \ldots, V[i+1]-1$.



Znázornění polí seznamu sousedů

Na tuto reprezentaci už stačí prostor $\Theta(n+m)$, což už je, na rozdíl od předchozího kvadratického prostoru, docela příjemné.

Pozorování: BFS s reprezentací seznamem sousedů běží v čase $\Theta(n+m)$.

Důkaz: Algoritmus zpracuje každý vrchol nejvýše jednou a stráví jím čas lineární v počtu odchozích hran, tedy $\Theta(\deg^-(v))$. Časová složitost celého algoritmu tedy činí:

$$\Theta\left(n + \sum_{v \in V(G)} \deg^-(v)\right) = \Theta(n+m).$$

♡

3. orákulum

Další možností reprezentace je jakési orákulum, které nám na požádání řekne (spočítá), kam vedou hrany z daného vrcholu. To se hodí například tehdy, pokud graf vznikl výpočtem a nechceme plýtvat pamětí na jeho uložení. To se hodí například u už zmíněného hlavolamu „Lloydova osmička".

Neorientované grafy

Chceme-li reprezentovat neorientovaný graf, uložíme každou hranu v obou orientacích.

Výpočet vzdáleností

Abychom mohli využít toho, že algoritmus prochází vrcholy grafu ve vlně, k výpočtu vzdáleností, doplníme do něj dvě pomocná pole: $D[v]$ bude říkat, na kolik kroků jsme se do v dostali, $P[v]$ bude obsahovat *předchůdce* vrcholu v, totiž vrchol u, ze kterého jsme se do v dostali po hraně a jehož $D[u] = D[v] - 1$.

Rozšířený algoritmus:

1. $Q \leftarrow \{v_0\}$.
2. $Z[*] \leftarrow 0, Z[v_0] \leftarrow 1$.
3. $D[*] \leftarrow \infty, D[v_0] \leftarrow 0$.
4. Dokud $Q \neq \emptyset$ opakujeme:
5. Vyzvedneme vrchol u z Q.
6. Pro každý vrchol v, do kterého vede hrana z vrcholu u:
7. Je-li $Z[v] = 0$:
8. $Z[v] \leftarrow 1, D[v] \leftarrow D[u] + 1, P[v] \leftarrow u$
9. Přidáme v do Q.

Definice: *Fáze běhu algoritmu*: Ve fázi F_0 je zpracováván vrchol v_0. Ve fázi F_{i+1} jsou zpracovávány vrcholy uložené do fronty Q během fáze F_i.

Pozorování: Každý vrchol v dosažitelný z v_0 se účastní právě jedné fáze, a to té s číslem $D[v]$.

Lemma: Po zastavení BFS pro všechny vrcholy dosažitelné z v_0 platí, že $D[v]$ je rovno $d(v_0, v)$, totiž vzdálenosti (délce nejkratší cesty) z v_0 do v.

Důkaz: Nejprve si uvědomíme, že kdykoliv je v označen, vede do něj z v_0 nějaký sled délky v (indukcí, stejně jako jsme před chvílí dokazovali, že BFS projde všechny dosažitelné vrcholy). Proto nemůže být $D[v]$ menší než $d(v_0, v)$.

Sporem dokážeme, že nemůže být ani větší. Předpokládejme, že existuje nějaký „špatný" vrchol v, pro který je $D[v] > d(v_0, v)$. Nechť P je některá z nejkratších cest

z v_0 do v. Z možných špatných vrcholů si vyberme takový, jehož P je nejkratší možná. Jelikož pro vrchol v_0 je zajisté $D[v_0] = d(v_0, v_0) = 0$, musí být v různý od v_0, takže má na P nějakého předchůdce u. Pro toho ovšem je vzdálenost spočítána správně: $D[u] = d(v_0, u) = d(v_0, v) - 1$.

Uvažujme nyní, co se stalo v okamžiku, kdy jsme $D[u]$ nastavili. Tehdy jsme u uložili do fronty, po čase jsme ho z fronty zase vytáhli a prozkoumali jsme všechny vrcholy, do níž vede z u hrana. Tedy i vrchol v, takže $D[v]$ v tomto okamžiku nemůže být větší než $D[u] + 1 = d(v_0, v)$, a to je spor. ♡

Víme tedy, že BFS správně spočítá délky nejkratších cest do všech vrcholů grafu. Pomocí předchůdců v poli P můžeme tyto cesty dokonce snadno rekonstruovat: předposledním vrcholem na nejkratší cestě do vrcholu v musí být vrchol $P[v]$, jeho předchůdcem $P[P[v]]$, ..., až do vrcholu v_0.

Předchůdci nám tedy kódují strukturu nejkratších cest do všech vrcholů. Můžeme se na ně dívat také následovně:

Definice: Strom nejkratších cest je orientovaný strom s množinou vrcholů $W = \{v \in V(G) \mid v \text{ dosažitelný z } v_0\}$ a hranami $\{(P(v), v) \mid v \in W, v \neq v_0\}$.

Pozorování: Kořenem stromu nejkratších cest je vrchol v_0, cesta v tomto stromu z v_0 do v (jednoznačně určená, je to strom) je pak jednou z nejkratších cest z v_0 do v v původním grafu.

Komponenty souvislosti

V neorientovaných grafech můžeme BFS jednoduše použít na nalezení komponent souvislosti. Již víme, že BFS spuštěné z vrcholu v_0 projde právě ty vrcholy, které jsou z v_0 dosažitelné, což jsou v neorientovaném grafu přesně ty, které leží v téže komponentě.

Stačí opakovaně spouštět BFS z dosud neoznačených vrcholů, pokaždé nám označí jednu komponentu.

Algoritmus:

1. Pro všechny vrcholy $v \in V(G)$ opakujeme:
2. Pokud je vrchol v neoznačený:
3. Spustíme $BFS(v)$ a přiřadíme objevené vrcholy nové komponentě.

To, co jsme o BFS zjistili, můžeme shrnout do následující věty:

Věta: $BFS(v_0)$ v čase $\Theta(n + m)$ spočte:

- vrcholy dosažitelné z v_0
- vzdálenosti těchto vrcholů od v_0
- strom nejkratších cest z v_0

Prohledávání do šířky ale není jediný algoritmus, který nějak systematicky prochází graf. Jak už název kapitoly napovídá, budeme se zabývat ještě druhým algoritmem, prohledáváním do hloubky. Podívejme se, jak bude vypadat ...

Prohledávání do hloubky *Depth-First Search – DFS*

Tento algoritmus neprochází graf ve vlně jako BFS, nýbrž rekurzivně. Vždy se

zanoří co nejhlouběji až do listu a pak se o kus vrátí a opět se snaží zanořit. Vrcholy, ve kterých už byl, ignoruje.

Opět uvažme nejdříve graf orientovaný. Následně si ukážeme, že v neorientovaném grafu budou pouze malé změny.

Budeme používat podobné značení jako u BFS. V poli Z si budeme pamatovat, zda jsme vrchol již navštívili (hodnota 1), nebo ne (hodnota 0). Aby se nám algoritmus lépe analyzoval, zavedeme proměnnou T, která bude fungovat jako hodiny – v každém kroku algoritmu se zvětší o jedničku. Do polí *in* a *out* si budeme ukládat čas prvního a posledního průchodu vrcholem.

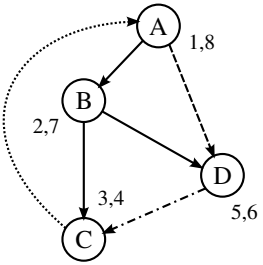
Algoritmus:

1. Inicializace: $Z[*] \leftarrow 0, T \leftarrow 1$, $in[*] \leftarrow ?$, $out[*] \leftarrow ?$
2. $DFS(v)$:
3. $\quad Z[v] \leftarrow 1$, $in[v] \leftarrow T$, $T \leftarrow T+1$
4. $\quad$ Pro w: $vw \in E(G)$:
5. $\quad\quad$ Pokud $Z[w] = 0$, zavoláme $DFS(w)$
6. $\quad out[v] \leftarrow T$, $T \leftarrow T+1$

Věta: DFS(v_0) v čase $\Theta(m+n)$ označí právě všechny vrcholy dosažitelné z v_0.
Důkaz: Korektnost dokážeme stejným argumentem, jako u BFS.

V analýze časové složitosti si pak opět uvědomíme, že algoritmus zavoláme na každý vrchol nejvýše jednou (pak už je označený) a zpracováním vrcholu strávíme čas lineární v počtu hran, které z něj vedou. Celkem tedy prozkoumáme každou hranu nejvýše jednou a strávíme tím konstantní čas. ♡

Vyzkoušejme si DFS na grafu z následujícího obrázku:



Znázornění průběhu DFS a typů hran

Graf je reprezentován tak, že v každém vrcholu jsou hrany uspořádány zleva doprava. Čísla u vrcholů ukazují jejich *in* a *out*.

Můžeme si všimnout, že k různým hranám se DFS chová různě. Po některých projde, jiné vedou do už prozkoumaných vrcholů, ale někdy do takových, ze kterých jsme se ještě nevrátili, jindy do už zpracovaných. Za chvíli uvidíme, že mohou nastat čtyři různé situace. Nejsnáze se poznávají podle hodnot *in* a *out*.

Pozorování: Chod algoritmu můžeme popsat posloupností závorek: $(_v$ bude značit „vstoupili jsme do vrcholu v“ (a nastavili $in(v)$), $)_v$ budiž opuštění vrcholu (nastavení

$out(v)$). Tato posloupnost bude správně uzávorkovaná, čili páry závorek se nebudou křížit.

Klasifikace hran: DFS dělí hrany na následující čtyři druhy. Hrana uv je:

- *Stromová* (na našem obrázku plná) pokud po ní DFS prošlo do neoznačeného vrcholu. Všimněte si, že stromové hrany společně tvoří strom orientovaný od kořene v_0. (Je to strom, jelikož vzniká postupným přidáváním listů.) Tomuto stromu se říká DFS strom.
- *Zpětná* (tečkovaná) – taková hrana vede do vrcholu, do kterého jsme vstoupili, ale ještě jsme ho neopustili (to je vrchol, který máme při rekurzi na zásobníku). Odpovídá uzávorkování $(_v \ldots (_u \ldots)_u \ldots)_v$.
- *Dopředná* (čárkovaná) – vede do vrcholu, který jsme už opustili, ale který je potomkem aktuálního vrcholu: $(_u \ldots (_v \ldots)_v \ldots)_u$.
- *Příčná* (čerchovaná) – vede do vrcholu, který jsme už opustili, ale který ve stromu neleží pod aktuálním vrcholem. Musí tedy vést do vrcholů „nalevo“ od stromové cesty z v_0 do u. Napravo totiž leží vrcholy, které v okamžiku opouštění u ještě nebyly prozkoumané, takže hrana uv by se stala stromovou. Příčné hrany poznáme podle uzávorkování $(_v \ldots)_v \ldots (_u \ldots)_u$.
- Jiné druhy hran nemohou existovat, probrali jsme totiž všechny možnosti, v jakém vztahu mohou být páry závorek $(_u)_u$ a $(_v)_v$.

Kterého typu hrana je, můžeme tedy poznat podle značky $Z[v]$ a podle hodnot *in* a *out* vrcholů u a v.

Neorientované grafy: V neorientovaných grafech (každou hranu vidíme jako dvě orientované hrany) je situace daleko jednodušší. Buďto hranu objevíme jako stromovou (a v opačném směru ji vidíme jako zpětnou), nebo ji objevíme jako zpětnou (a v opačném směru se jeví dopřednou). Příčné hrany nemohou existovat, protože k nim opačná hrana by byla příčná vedoucí zleva doprava, což víme, že nenastane.