Typy hran ($v \to w$):

- Stromové hrany ... po nich DFS prošlo. $\{(A \to B), (B \to C), (B \to D)\}$
- Zpětné hrany $<<>_v>_w$... vedou do předchůdce v ve stromu. $\{(C \to A)\}$
- Dopředné hrany $<<>_w>_v$... vedou do potomka. v $\{(A \to D)\}$
- Příčné hrany $<>_w<>_v$... vedou do vrcholu v v sousedním podstromě, vždy zprava doleva. $\{(D \to A)\}$

Pozorování: Hrany, po kterých DFS prošlo, tvoří DFS strom.

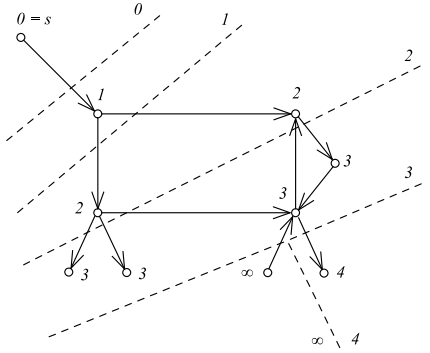
Pozorování: Intervaly $(in(v), out(v))$ $\forall v \in V(G)$ tvoří dobré uzávorkování. (Intervaly synů disjunktně vyplňují otce $\Rightarrow$ intervaly se nemohou křížit).

Nakonec si ještě uvědomme, jak bude vypadat prohledávání do hloubky na neorientovaném grafu. Algortimus bude úplně stejný, jenom se nám zredukuje počet typů hran na dvě: *stromové* a *zpětné*. Ani *dopředné* ani *příčné* nebudou existovat, neboť se z *příčných* stanou *stromové* a z *dopředných zpětné*.

3. Prohledání do šířky a do hloubky *()*

Prohledání do šířky (BFS) *Breadth-First Search*

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 nejkraší cesty mezi dvěma vrcholy v grafu.



Prasečí graf

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, či nikoli. Pro vrchol v_0 si tedy dosazením jedničky zapamatujeme, že je již navštívený. V dalším kroku pak zkoumáme frontu Q: pokud není prázdná, vezmeme z ní první vrchol a podíváme se na všechny jeho sousedy w. Pokud ještě nejsou označené (tedy $Z[w] = 0$), tak je označíme (zapamatujeme si, že je předáváme ke zpracování a už je nemáme znovu navštěvovat) a přidáme je do fronty k následnému zpracování. Takto cyklus opakujeme, dokud není fronta prázdná.

Poznámka: Nejprve se podívejme, jak algoritmus pracuje na orientovaném grafu. Pro neorientovaný graf funguje algoritmus analogicky, což si ukážeme později.

Algoritmus:

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

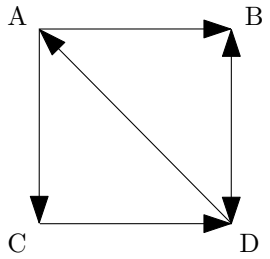
První krok indukce je triviální, neboť cesta z v_0 do v_0 existuje vždy. Nyní si představme, že označujeme vrchol v přes hranu uv. To znamená, že vrchol u již musel být označený. Dle indukčního předpokladu tedy existuje cesta z v_0 do u, a tudíž pokud k této cestě „přilepíme“ hranu uv, dostáváme sled 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. Museli jsme tedy použít hranu $v_{i-1}\ v_i$ a označit vrchol v_i, což je SPOR. ♡

Nyní tedy víme, že je algoritmus 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

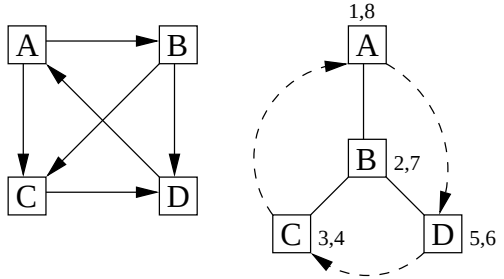
Označme vrcholy grafu na následujícím obrázku písmeny A, B, C, D. Pokud bychom chtěli tento graf uchovat v paměti počítače, máme na výběr hned několik způsobů, jak to udělat.



1. matice sousednosti

Matice sousednosti pro graf G na n vrcholech je čtvercové pole A o velikosti $n \times n$, jehož prvky na souřadnicích i, j jsou dány následujícím předpisem:

$$A_{i,j} = \begin{cases} 1 \Leftrightarrow \{i,j\} \in E \\ 0 \Leftrightarrow \{i,j\} \notin E \end{cases}$$



Graf a znázornění průběhu DFS s jednotlivými hranami:

Můžeme si všimnout, že jak DFS prochází graf, rozděluje hrany do 4 skupin: hrany *stromové, zpětné, dopředné a příčné*.

Jedině *stromové* hrany jsou takové, že se po nich DFS opravdu vydá. Vedou totiž do vrcholu, který nebyl dosud objeven. V ukázkovém grafu to jsou hrany: $\{(A \to B), (B \to C), (B \to D)\}$.

Pokud algoritmus objeví z vrcholu v hranu do již dříve navštíveného vrcholu w a zároveň platí, že w je ve stejném podstromu jako v, tak nazveme hranu vw *zpětnou*. Pro rozpoznání je důležité, že vrchol w byl již objeven, ale ještě ne opuštěn. V ukázkovém grafu se jedná o hranu: $(C \to A)$.

Když při prohledávání sousedů vrcholu v narazíme na vrchol w, který jsme již navštívili, a to v podstromě vrcholu v, tak nazveme hranu vw *dopřednou*, neboť vede z v do jeho potomka. Platí tedy, že jsme nejdříve objevili vrchol v, potom vrchol w, pak jsme vrchol w opustili a nyní jsme na něj znovu narazili po dopředné hraně. V ukázkovém grafu hrana: $(A \to D)$.

Posledním typem hran je *příčná* hrana. Ta vede do vrcholu v sousedním podstromě zprava doleva. V tomto případě jsme tedy nejdříve objevili vrchol w, ten jsme následně opustili a až pak jsme objevili vrchol v. V ukázkovém grafu to je hrana: $(D \to C)$.

K zamyšlení: Proč nemohou vést příčné hrany také zleva doprava?

K rozpoznávání typů hran se nám tedy velmi hodí pole *in* a *out*, ve kterých si pamatujeme časy objevení a opuštění vrcholu. Podle toho, jak se intervaly objevení a opuštění obou vrcholů překrývají, můžeme jednoznačně rozhodnout, o jaký typ hrany se jedná:

U zpětných hran je pořadí: $in(w)$, $in(v)$, $out(v)$, $out(w)$. Intervaly do sebe budou zanořené takto: $<<>_v>_w$.

U dopředných hran je pořadí: $in(v)$, $in(w)$, $out(w)$, $out(v)$. Intervaly do sebe budou zanořené takto: $<<>_w>_v$.

U příčných hran je pořadí: $in(w)$, $out(w)$, $in(v)$, $out(v)$. Intervaly do sebe budou zanořené takto: $<>_w<>_v$.

Pozn: Používáme zde toto značení: $<>_v=< in(v), out(v) >$. Jedná se o interval objevení a opuštění vrcholu v.

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

Poznámka: Algoritmus na prohledávání do šířky bude fungovat i na neorientovaném grafu. Můžeme si jednoduše představit, že je to orientovaný graf, kde každá hrana má oboustrannou orientaci.

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 (DFS) *Depth-First Search*

Tento algoritmus neprochází graf ve vlně jako BFS, ale prochází ho 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). Navíc proměnná T bude značit dobu běhu algoritmu - tedy jakési „hodiny“. Při každém nalezení nového vrcholu či jeho opuštění tuto proměnnou zvýšíme o 1. Do polí *in* a *out* si budeme ukládat čas (prvního) nalezení a opuštění vrcholu.

Algoritmus:

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

Věta: DFS(v_0) v čase $\Theta(m+n)$ označí právě všechny vrcholy dosažitelné z v_0.

Důkaz: Nejdříve je potřeba dokázat, že pokud je vrchol v dosažitelný z vrcholu v_0, tak jej DFS označí. Důkaz bude podobný jako u BFS.

V analýze časové složitosti si pak opět uvědomíme, že algoritmus vezme každý vrchol i hranu do ruky právě jednou, takže časová složitost bude $\Theta(n+m)$. ♡

Na pozicích i, j je jednička, pokud v grafu G vede hrana z vrcholu i do vrcholu j, jinak to je nula. 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. 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: $\Theta(n^2)$.

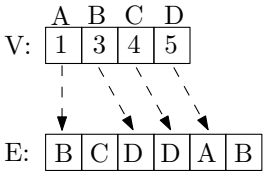
Důkaz: Už jsme si uvědomili, že každý vrchol se dostane do fronty Q nejvýše jednou. Pro každý vrchol ve frontě potřebujeme projít jeho sousedy, což nám trvá s reprezentací maticí sousednosti $\Theta(n)$. Vrcholů je celkem n, tedy časová složitost je $\Theta(n^2)$. ♡

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. K sousedům vrcholu i se tedy dostaneme již snadno - nalezneme je na pozicích $V[i]$ ... $V[i+1]-1$.



Znázornění polí seznamu sousedů.

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

Pozorování: BFS běží v čase: $\Theta(n+m)$.

Důkaz: Algoritmus vezme každý vrchol i každou hranu do ruky nejvýše jednou. Časová složitost bude tedy:

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

♡

3. orákulum

Další možností reprezentace je pak jakési orákulum, které nám řekne (spočítá), kam vedou hrany z daného vrcholu... To se hodí například tehdy, pokud si nepotřebujeme pamatovat celý graf, ale stačí nám naleznout sousedy nějakého vrcholu, které orákulum jednoznačně výpočtem dokáže určit. Například při řešení známé úlohy odměření daného množství vody za pomocí nádob různých objemů můžeme tento způsob reprezentace grafu použít. Problém převedeme na prohledávání grafu do šířky, kde vrcholům odpovídají jednotlivé stavy. Stav říká, kolik vody je zrovna ve které nádobě. Potom nám již stačí ze zadaného počátečního vrcholu (stavu) najít cestu do cílového. Orákulum je zde vlastně funkce, které předáme vrchol a ona nám vrátí všechny vrcholy sousední – tedy takové stavy, ke kterým dojde, když vyzkouší všechny možnosti přelití kapaliny z jedné nádoby do druhé.

Rozšíření algoritmu:

Abychom mohli využít toho, že algoritmus prochází vrcholy grafu ve vlně, a jiných hezkých vlastností, tak si dodefinujeme následující označení:

V poli D bude pro každý vrchol uložena vzdálenost od počátečního vrcholu. V poli P si budeme pro každý vrchol pamatovat jeho předchůdce. Dále budeme využívat fáze běhu algoritmu, které budou simulovat onu vlnu:

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.

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 w, který je sousedem vrcholu u:
7. Je-li $Z[w] = 0 \Rightarrow Z[w] \leftarrow 1, D[w] \leftarrow D[u]+1, P[w] \leftarrow u$
8. Přidáme w do Q.

Lemma: Na konci BFS pro všechny vrcholy dosažitelné z v_0 platí, že vrchol v byl zpracován ve fázi F_i právě tehdy, když vzdálenost v_0 a v (délka nejkratší cesty z v_0 do v) je rovna i. Formálně zapsáno: $v \in F_i \Leftrightarrow d(v_0, v) = i$. (Kde $d(x,y)$ je délka nejkratší cesty z x do y).

Důkaz: „$\Longrightarrow$“: Důkaz provedeme indukcí podle i (čísla fáze běhu algoritmu).

První krok indukce je triviální, neboť ve fázi F_0 je označen (dle definice) pouze vrchol v_0 a to je jediný vrchol vzdálený 0 od v_0.

Pokud je vrchol v zpracováván ve fázi F_i, pak musel být zařazen do fronty během fáze F_{i-1} jako soused nějakého vrcholu u. Pro vrchol u můžeme použít indukční předpoklad, tedy že délka nejkratší cesty z v_0 do u je $d(v_0, u) = i-1$. Pak tedy $d(v_0, v) \leq i$, neboť ještě nevíme, zda cesta $v_0, \ldots, u, v$ je nejkratší. Kdyby ale existovala nějaká kratší, tedy délky nejvýše $i-1$, tak by byl vrchol v objeven už dříve než ve fázi F_i. Proto $d(v_0, v) = i$.

„$\Longleftarrow$“: Každý dosažitelný vrchol padne do nějaké fáze (viz. minulé lemma). ♡

Již tedy víme, že vrchol v_i, jehož vzdálenost od vrcholu v_0 je i, bude zpracován v i-té fázi. Jak ale po skončení algoritmu zjistíme, ve které fázi byl zpracován, neboli jak je vzdálený od startovního vrcholu? Tato informace je právě uložena v poli D s indexem i (v $D[i]$).

Zároveň nás může zajímat, jak bychom nejkratší cestu z v_0 do v_i rekonstruovali. Pro tento účel jsme si zavedli pole P. Nejkratší cesta z v_0 do v_i bude v obráceném pořadí vypadat: $v_i, P[v_i], P[P[v_i]], P[P[P[v_i]]], \ldots, v_0$.

Pozorování: Pokud víme, že $v_0, v_1, \ldots, v_{k-1}, v_k$ je nejkratší cesta z v_0 do v_k, pak $v_0, v_1, \ldots, v_{k-1}$ je nejkratší cesta z v_0 do v_{k-1}. Neboli prefix nejkratší cesty je nejkratší cesta.

Důkaz: Kdyby existovala kratší cesta z v_0 do v_{k-1}, pak bychom mohli zkrátit i cestu z v_0 do v_k.

Pozorování: BFS u neorientovaného grafu projde celou komponentu souvislosti.

Důkaz: Víme, že BFS(v_0) označí v právě tehdy, když existuje cesta z v_0 do v. V neorientovaném grafu existuje cesta z v_0 do právě všech vrcholů, které jsou ve stejné komponentě souvislosti jako v_0. Pokud tedy spustíme BFS na v_0, tak se postupně projdou všechny vrcholy této komponenty souvislosti. ♡

Pozorování: Pokud BFS postupně spouštíme na dosud neobarvené vrcholy v neorientovaném grafu, nalezneme nakonec v čase $\Theta(n+m)$ všechny komponenty souvislosti.

Algoritmus:

1. Pro všechny vrcholy $v \in V(G)$ opakuj:
2. Pokud je vrchol v neobarvený $\Rightarrow$ *BFS(v)*.

Důkaz: Každým spuštěním na dosud neobarvený vrchol neorientovaného grafu obarvíme právě jednu komponentu souvislosti (tu, ve které je tento vrchol). Postupně projdeme všechny vrcholy od prvního k poslednímu a vždy pokud je vrchol neobarvený, spustíme na něj BFS. Tak nakonec obarvíme všechny komponenty souvislosti. Časová složitost bude stejná jako u samotného BFS, tedy $\Theta(n+m)$. ♡

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