Algoritmy

IAL 2021/2022

Grafové algoritmy

Bohuslav Křena

krena@fit.vutbr.cz

Co nás dnes čeká

- **Grafové algoritmy**
  - grafy a jejich reprezentace
  - průchody – do šířky a do hloubky
  - nejkratší cesta v grafu
  - problém obchodního cestujícího
- **Paralelní algoritmy**

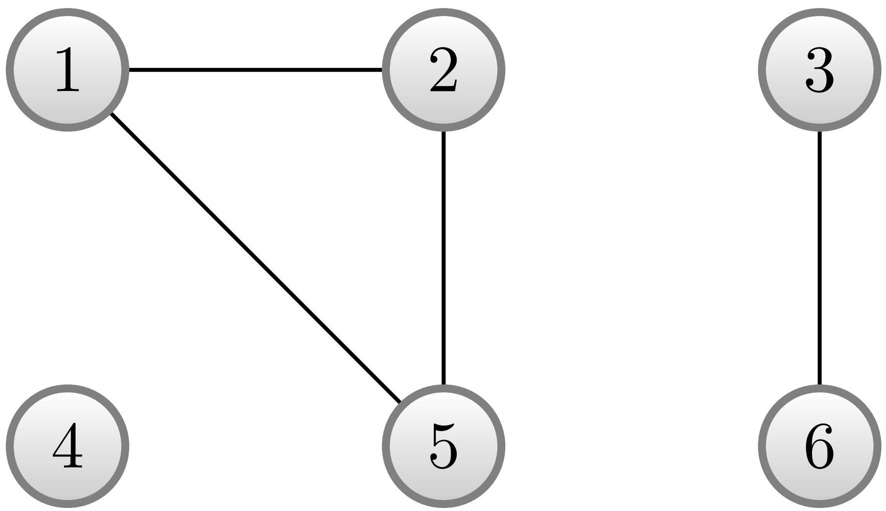
Část slidů byla s laskavým svolením autorů (Z. Křivka, T. Masopust) vytvořena na základě vybraných slidů z předmětu Grafové algoritmy (GAL).

Neorientovaný graf

Neorientovaný graf G je dvojice $G = (V, E)$, kde

- V je konečná množina **vrcholů** (uzlů) a
- E je množina **hran** tvaru $\{u, v\}$, kde $u, v \in V$ a $u \neq v$.
- Smyčky tedy nejsou povoleny.

Ukázka neorientovaného grafu:

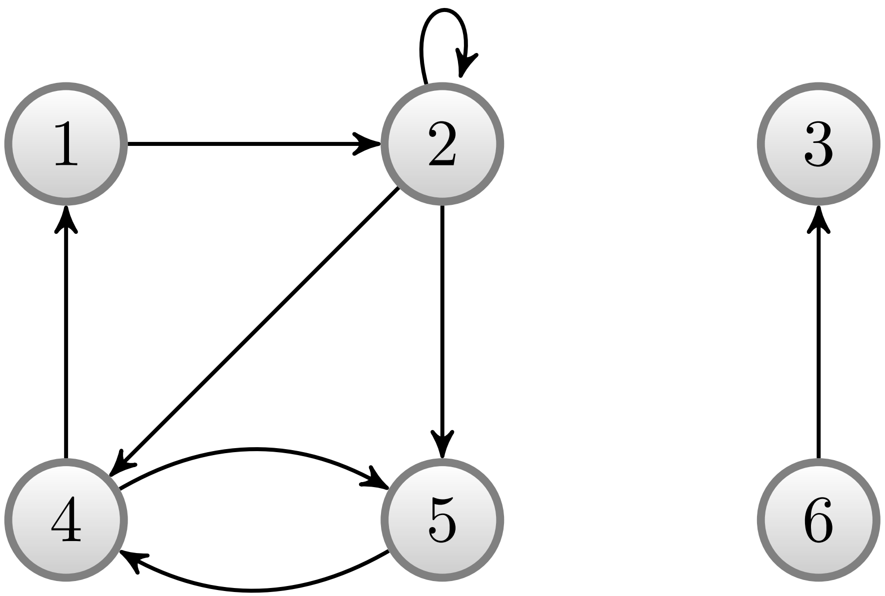


Orientovaný graf

Orientovaný graf G je dvojice $G = (V, E)$, kde

- V je konečná množina **vrcholů** a
- E je množina **hran** tvaru (u, v), kde $u, v \in V$.
- Hrana (u, u) se nazývá **smyčka**.

Ukázka orientovaného grafu:



Reprezentace grafů

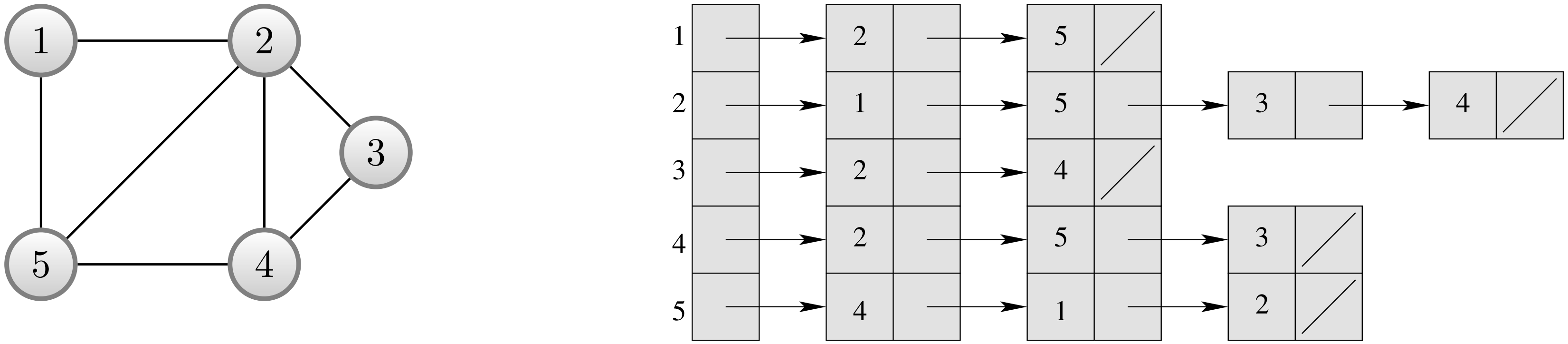
- Nechť $G = (V, E)$ je graf. Pak zavedeme následující značení:
    - $n = |V|$
    - $m = |E|$

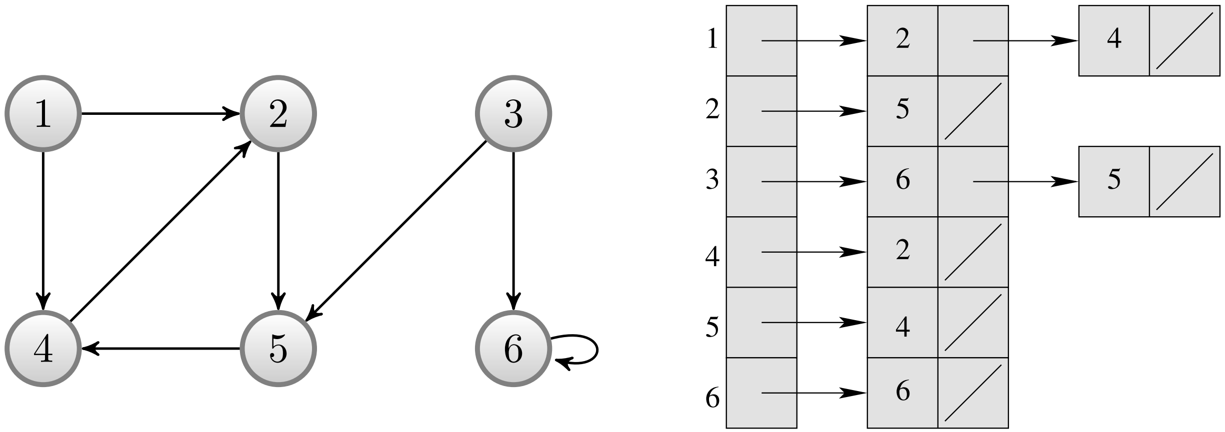
- **Seznam sousedů**
    - preferovaná reprezentace
    - výhodnější zejména pro řídké grafy, tj. takové grafy, kde $m \ll n^2$.
- **Matice sousednosti**
    - může být výhodnější pro husté grafy, tj. ty, kde m je skoro n^2,
    - nebo když potřebujeme rychle rozhodnout, zda graf obsahuje hranu spojující dva dané vrcholy.
- **Matice incidence**
    - Explicitně zachycuje hrany.

Seznam sousedů

Seznam sousedů grafu $G = (V, E)$ sestává z

- pole $Adj[1 \dots n]$ mající n seznamů, jeden pro každý vrchol z V,
- kde $Adj[u]$ obsahuje všechny vrcholy v takové, že $(u, v) \in E$.





Seznam sousedů – paměťová složitost

- Pokud G je orientovaný graf, pak součet délek seznamů seznamu sousedů je m, protože každá hrana tvaru (u, v) je reprezentována vrcholem v vyskytujícím se v $Adj[u]$.
- Pokud G je neorientovaný graf, pak součet délek seznamů seznamu sousedů je $2m$, protože každá hrana $\{u, v\}$ je reprezentována vrcholem v vyskytujícím se v $Adj[u]$ a vrcholem u vyskytujícím se v $Adj[v]$.
- Třída paměťové složitosti je v obou případech stejná, $\Theta(m + n)$.
- Nevýhoda: zjistit, zda je hrana (u, v) přítomná v grafu, znamená hledat vrchol v v celém seznamu $Adj[u]$.

Ohodnocený graf

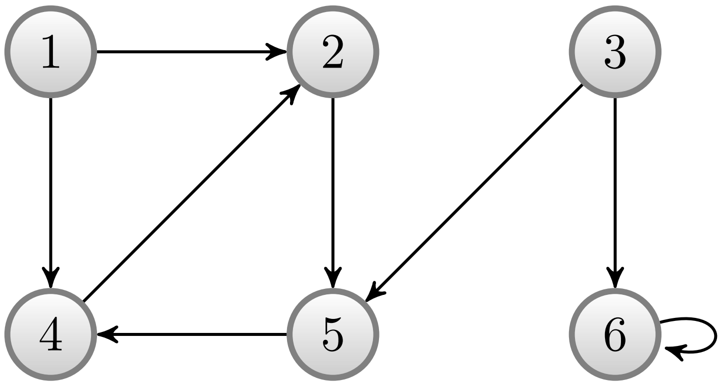
- Ohodnocený graf je takový graf, kde každá jeho hrana má přiřazenu nějakou hodnotu, typicky danou **váhovou funkcí** $w : E \rightarrow \mathbb{R}$.
- Seznam sousedů se snadno rozšíří pro ohodnocené grafy tak, že hodnota $w(u, v)$ je uložena s vrcholem v v seznamu $Adj[u]$.

Matice sousednosti

Nechť $G = (V, E)$ je graf a předpokládejme, že vrcholy jsou nějak očíslovány čísly $1, 2, \ldots, n$. Matice sousednosti $A = (a_{ij})$ je čtvercová matice řádu n taková, že

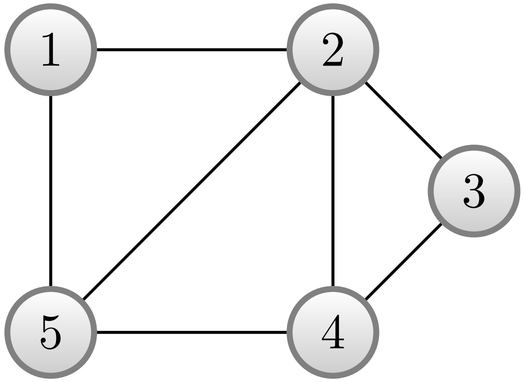
$$a_{ij} = \begin{cases} 1 & \text{pokud } (i, j) \in E, \\ 0 & \text{jinak.} \end{cases}$$

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



Matice sousednosti – paměťová složitost

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



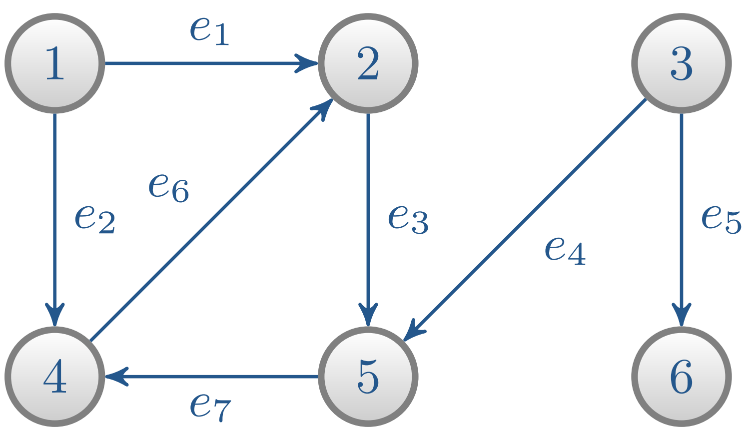
- Množství obsazené paměti je bez ohledu na počet hran $\Theta(n^2)$.
- Matice sousednosti neorientovaného grafu je symetrická. Stačí tedy do paměti uložit pouze část matice nad diagonálou.
- Nechť $G = (V, E)$ je ohodnocený graf, pak

$$a_{ij} = \begin{cases} w(i,j) & \text{pokud } (i,j) \in E, \\ NIL & \text{jinak,} \end{cases}$$

kde NIL je nějaká unikátní hodnota, většinou 0 či ∞.

Matice incidence

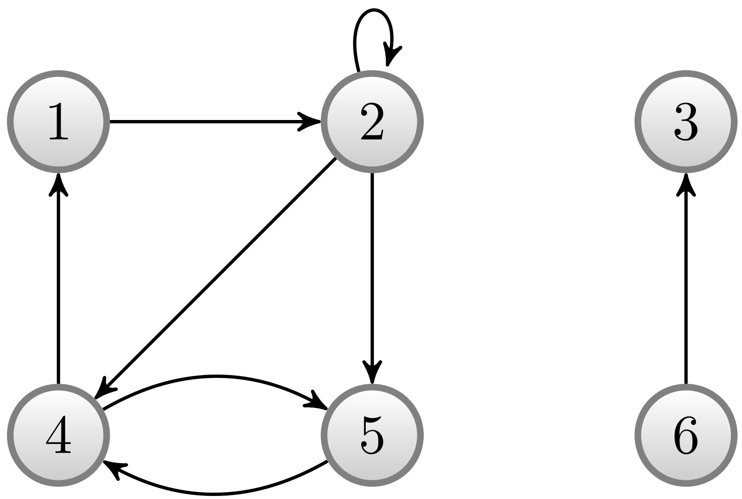
- Zachycuje incidenci vrcholů a hran:
  - $b_{ij} = 1 \quad \Rightarrow$ hrana j začíná ve vrcholu i
  - $b_{ij} = -1 \quad \Rightarrow$ hrana j končí ve vrcholu i
  - $b_{ij} = 2 \quad \Rightarrow$ hrana j je smyčkou u vrcholu i
  - $b_{ij} = 0 \quad \Rightarrow$ hrana j nemá s vrcholem i *nic společného*
- K hranám lze snadno přidat další informace.
- Množství obsazené paměti je $\Theta(n \times m)$.
  - Vhodné pro grafy s malým počtem hran.



	e_1	e_2	e_3	e_4	e_5	e_6	e_7
1	1	1	0	0	0	0	0
2	-1	0	1	0	0	-1	0
3	0	0	0	1	1	0	0
4	0	-1	0	0	0	1	-1
5	0	0	-1	-1	0	0	1
6	0	0	0	0	-1	0	0

Sled, tah, cesta

- Posloupnost vrcholů $\langle v_0, v_1, v_2, \ldots, v_k\rangle$, kde $(v_{i-1}, v_i) \in E$ pro $i = 1, 2, \ldots, k$, se nazývá **sled** délky k z v_0 do v_k.
- Sled je tedy určen vcholy $v_0, v_1, \ldots, v_k$ a hranami (v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k). Délka sledu je počet jeho hran.
- Pokud existuje sled s z u do u', říkáme, že u' je **dosažitelný** z u sledem s, značeno $u \overset{s}{\rightsquigarrow} u'$.
- **Tah** je sled, ve kterém se neopakují hrany.
- **Cesta** je sled, ve kterém se neopakují vrcholy.
- Uzavřená cesta se nazývá **kružnice**.
- Orientovaná kružnice se nazývá **cyklus**.
- Příklady
  - Sled $\langle 1, 2, 5, 4\rangle$ je tahem i cestou.
  - $\langle 2, 5, 4, 5\rangle$ je sled, který není cestou.
  - Je $\langle 2, 5, 4, 5\rangle$ tahem?



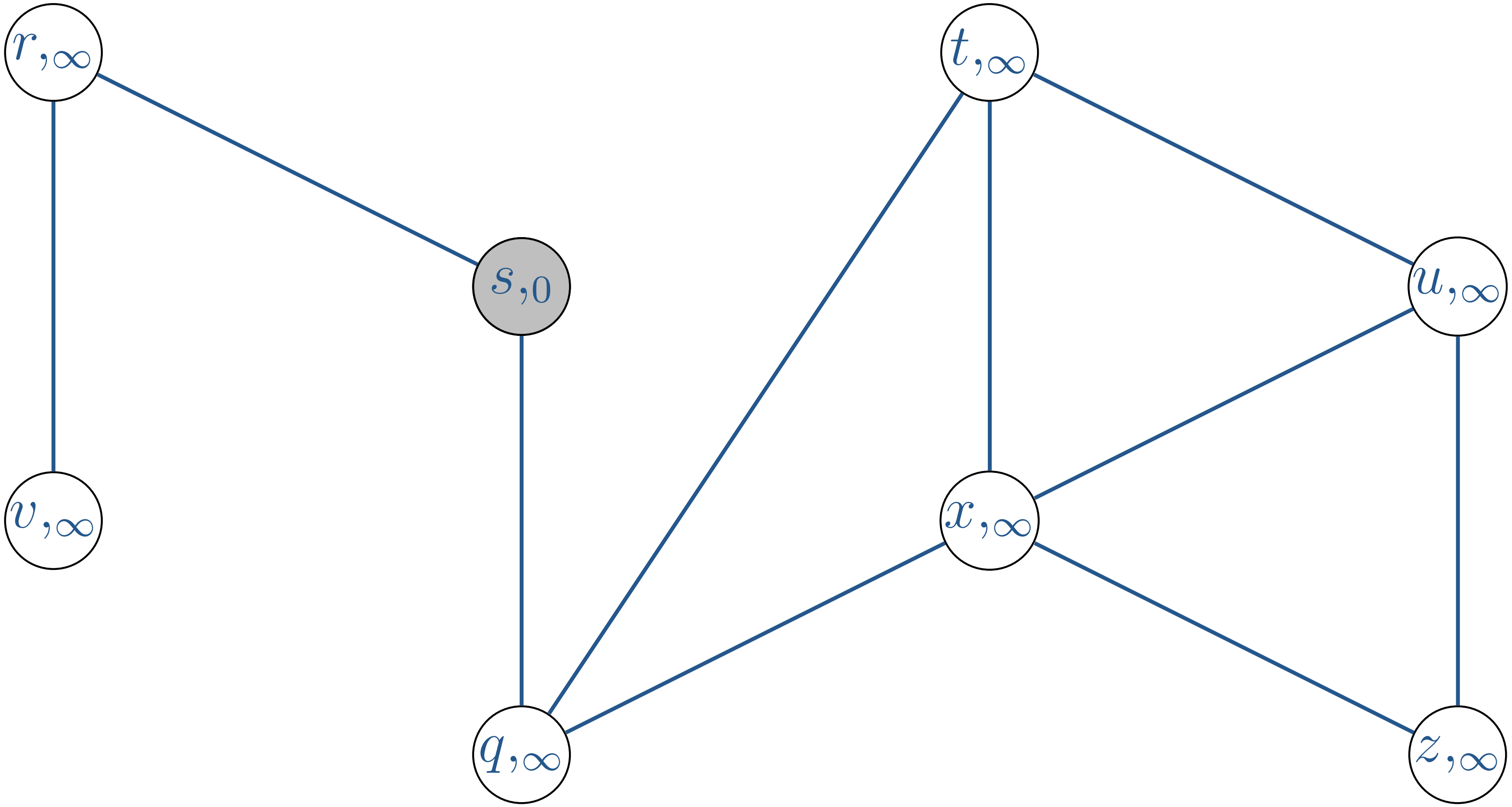
Prohledávání grafu do šířky

- Anglicky *Breadth-First Search (BFS)*
- Vstup: (ne)orientovaný graf $G = (V, E)$ a vrchol $s \in V$.
- Prochází všechny vrcholy dostupné z s a počítá jejich vzdálenost (počet hran) od s.
- Vytváří **strom prohledávání do šířky** s kořenem s obsahující všechny vrcholy dosažitelné z s. Cesta z s do v je nejkratší cestou v grafu.
- Při procházení grafu obarvuje vrcholy bílou, šedou a černou barvou.
- Reprezentace grafu – seznam sousedů.
- $color[u] \in \{$WHITE, GREY, BLACK$\}$.
- $\pi[u]$ je předchůdce u na cestě z s.
- $d[u]$ je vzdálenost (počet hran) u od s.

BFS(G, s)

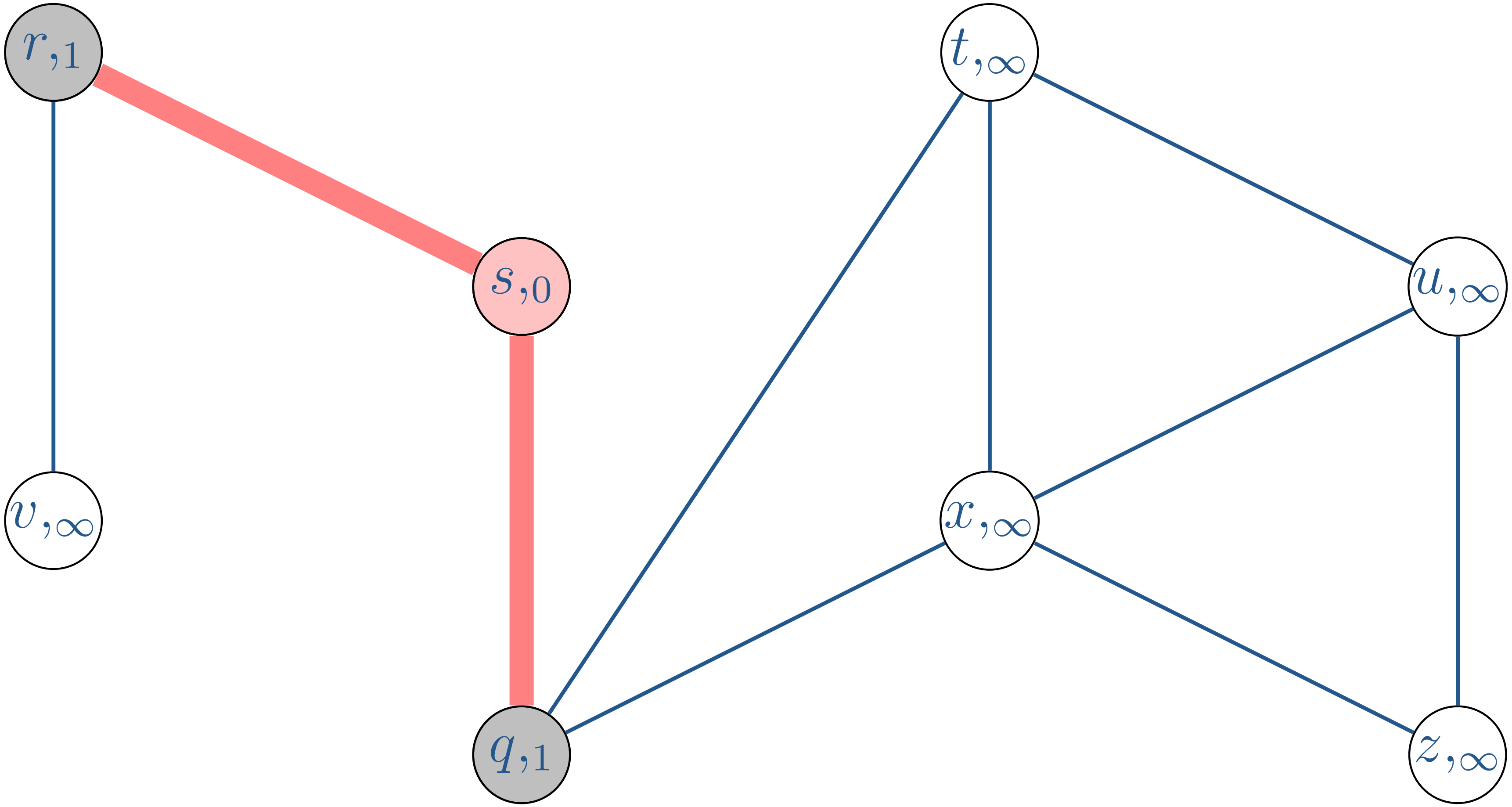
```
BFS(G, s)
color[s] ← GREY
d[s] ← 0
π[s] ← NIL
for KAŽDÝ VRCHOL u ∈ V − {s}
    do color[u] ← WHITE
       d[u] ← ∞
       π[u] ← NIL
InitQueue(Q)
Add(Q, s)
while NOT IsEmpty(Q)
    do u ← Front(Q)
       for KAŽDÝ v ∈ Adj[u]
           do if color[v] = WHITE
                 then color[v] ← GREY
                      d[v] ← d[u] + 1
                      π[v] ← u
                      Add(Q, v)
       Remove(Q)
       color[u] ← BLACK
```

BFS – příklad



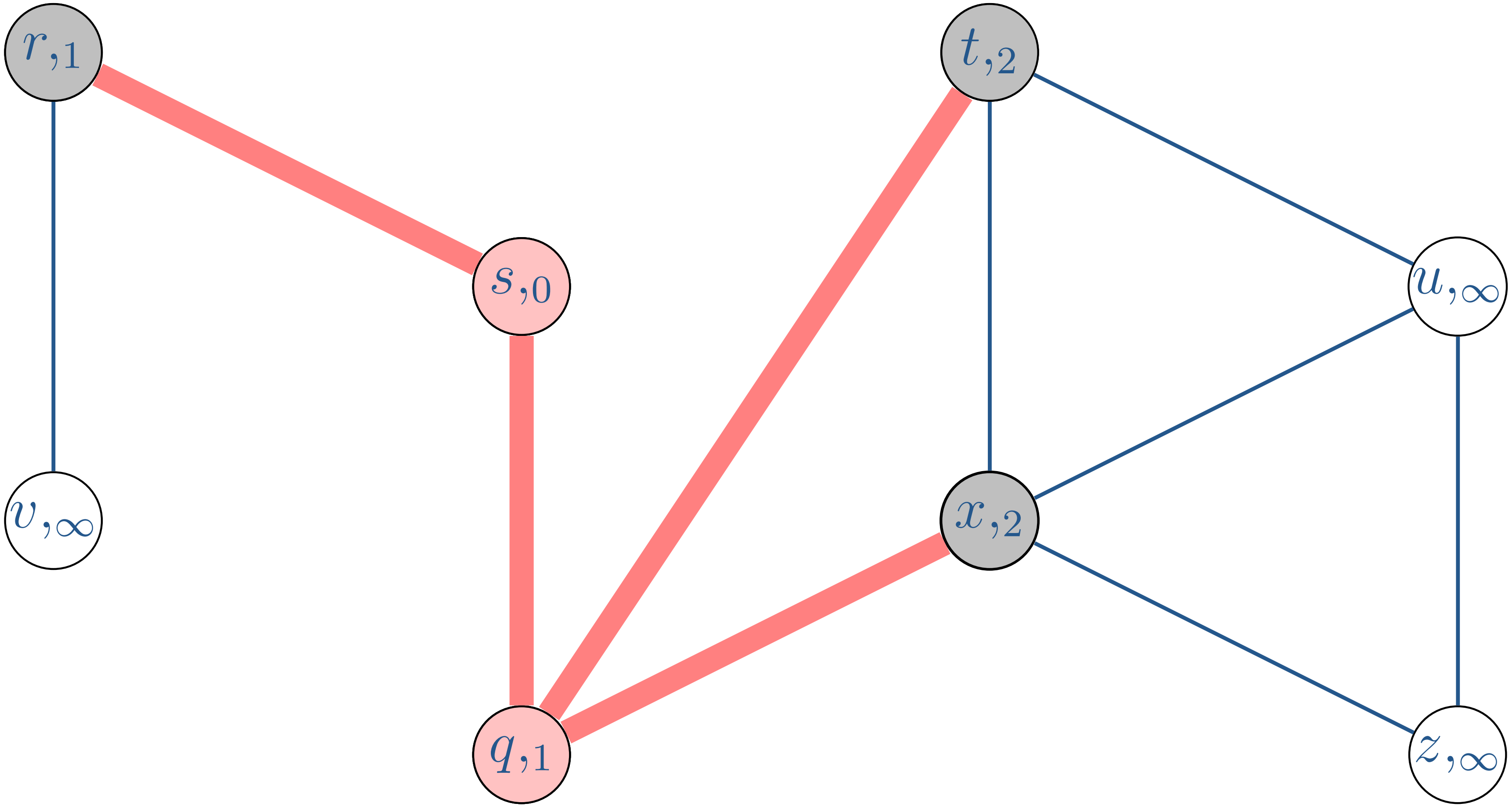
$Q = s$

BFS – příklad



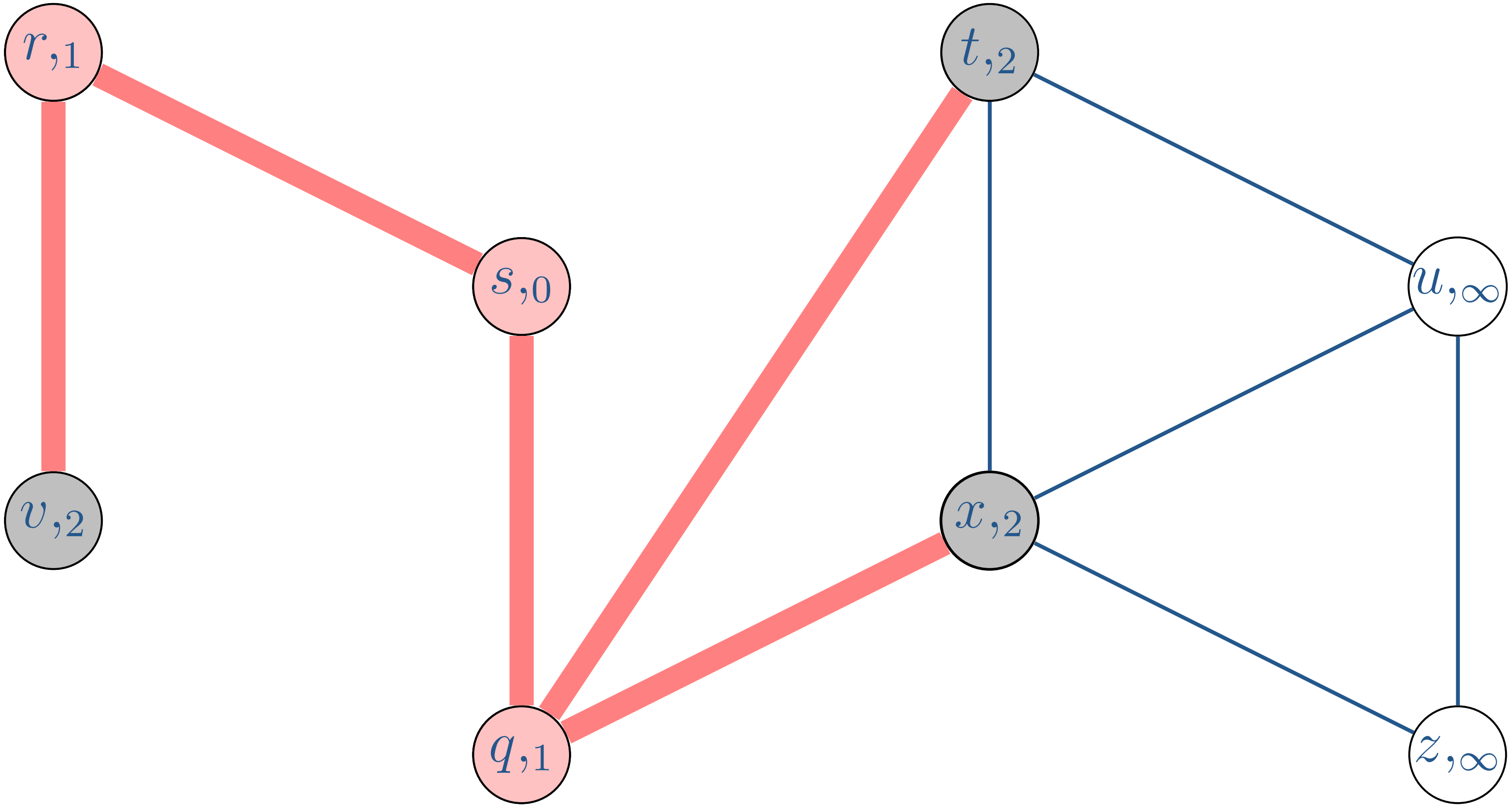
$Q = qr$

BFS – příklad



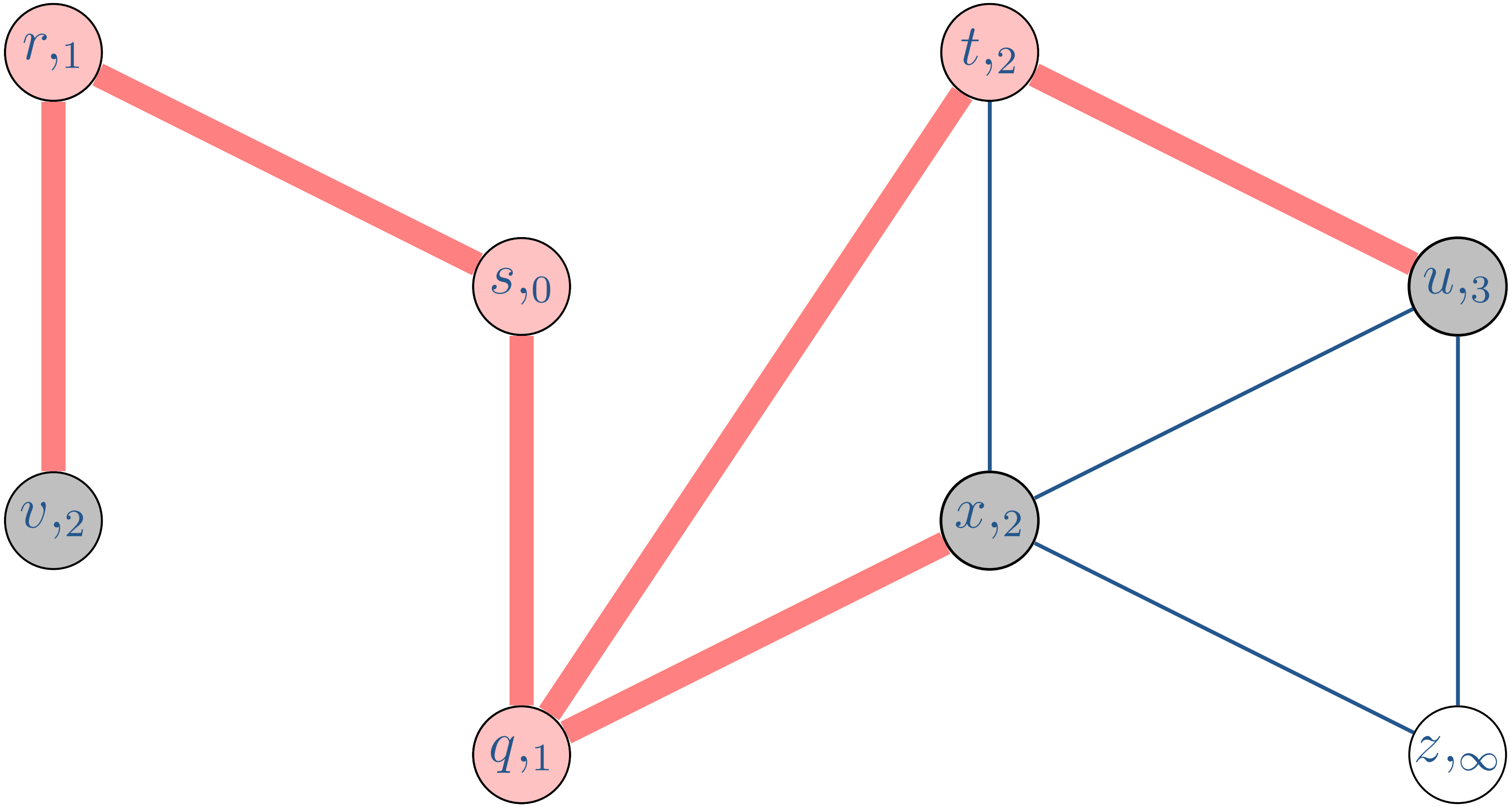
$Q = rtx$

BFS – příklad



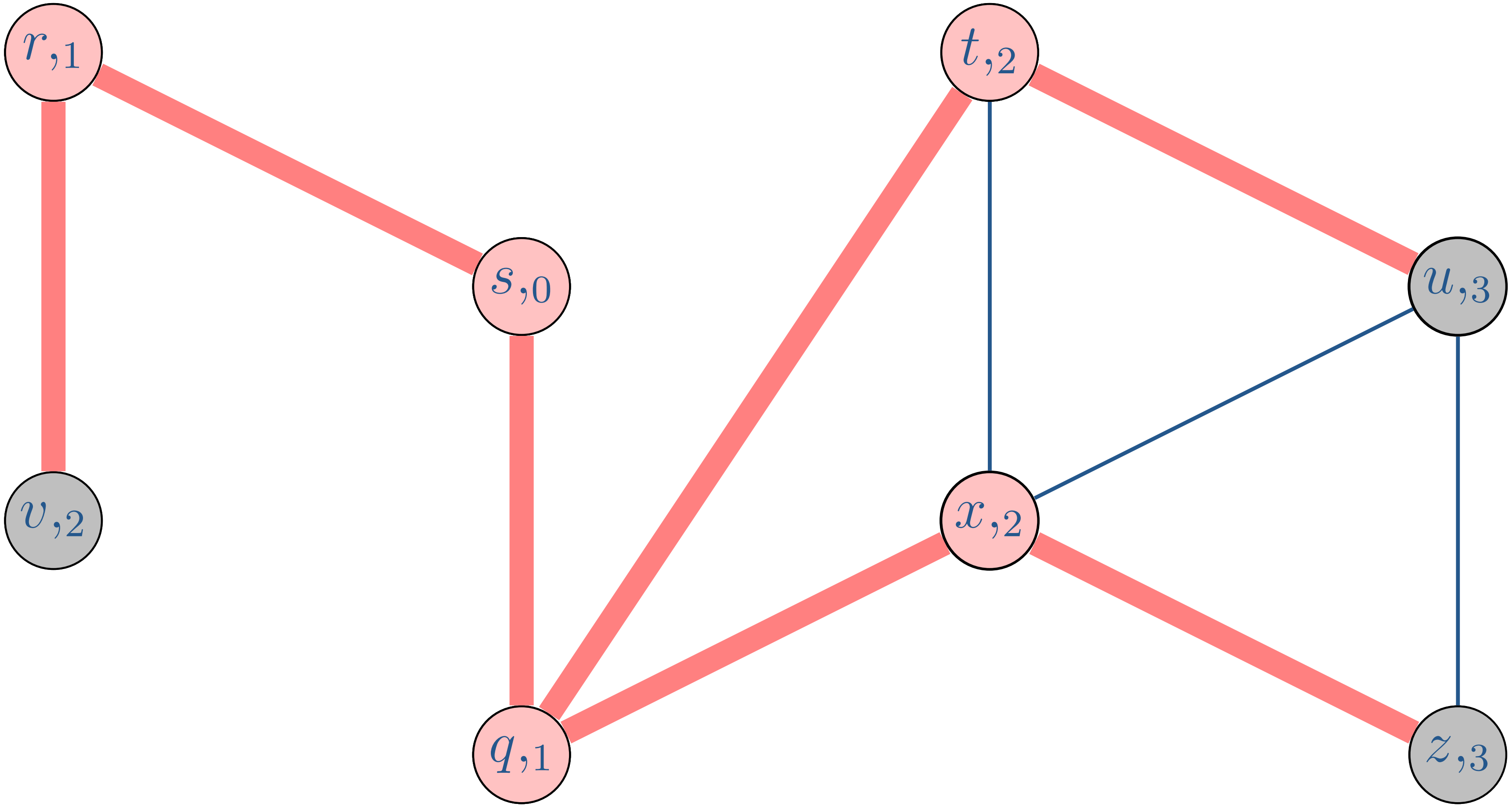
$Q = txv$

BFS – příklad



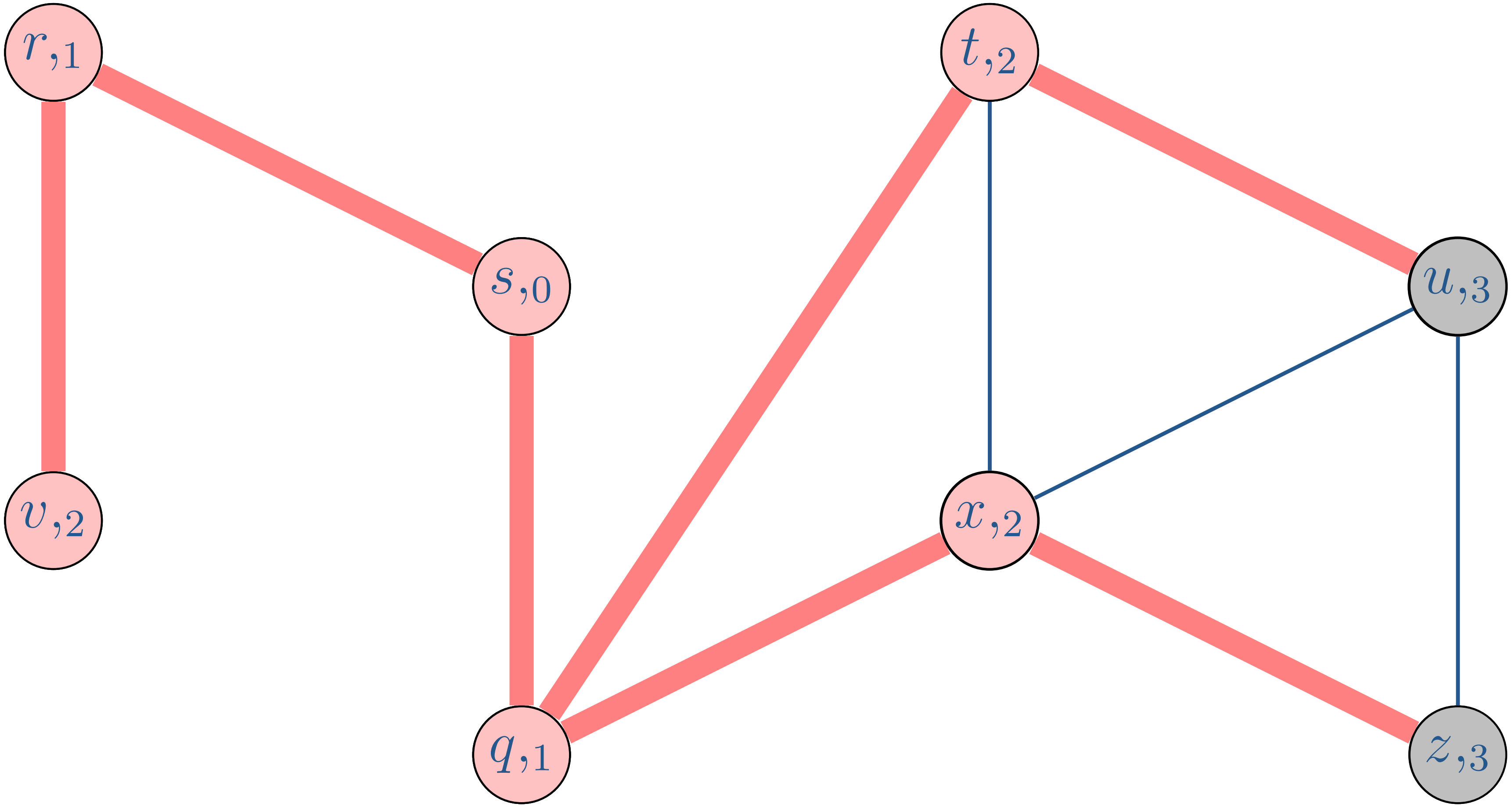
$Q = xvu$

BFS – příklad



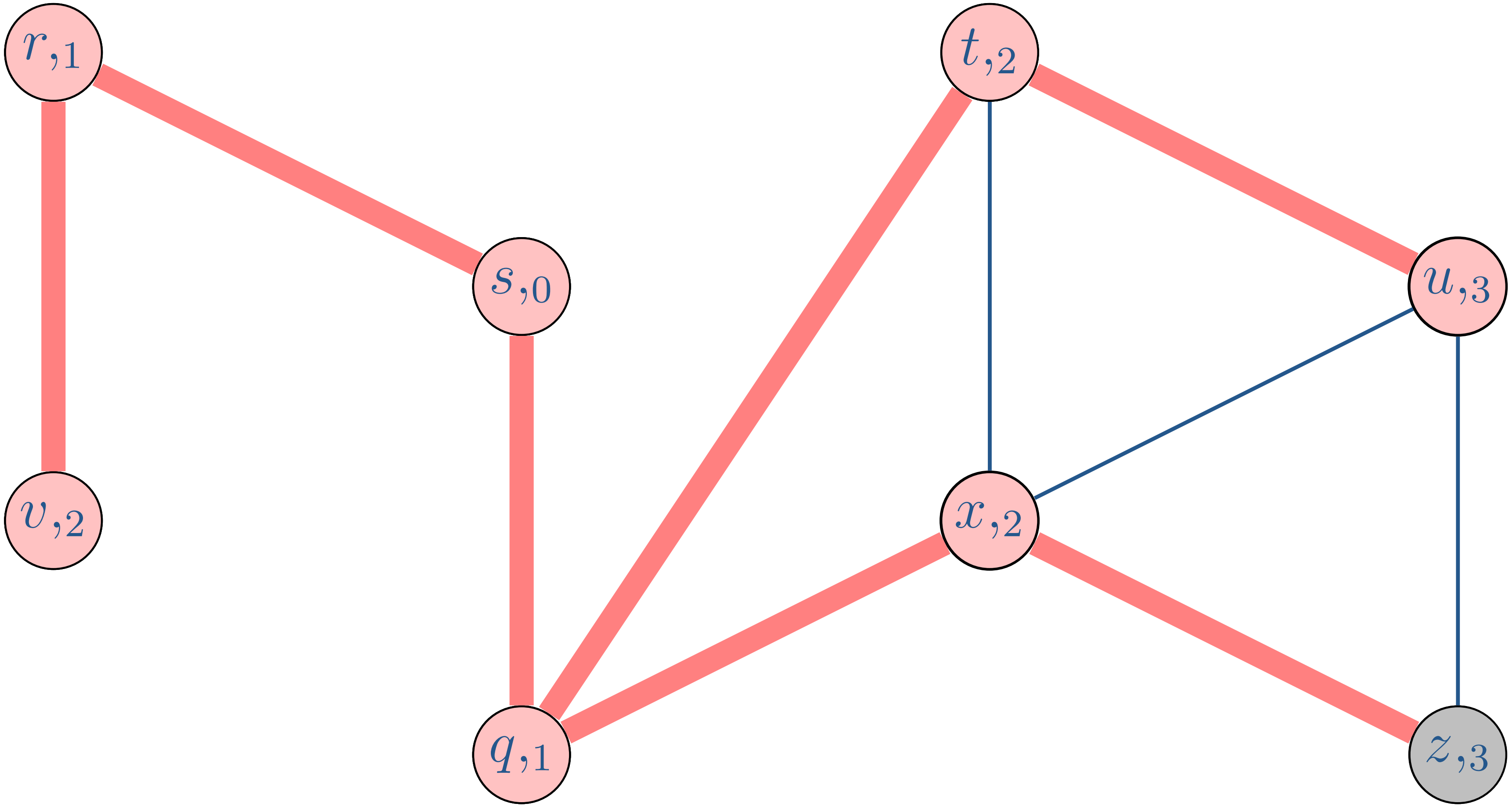
$Q = vuz$

BFS – příklad



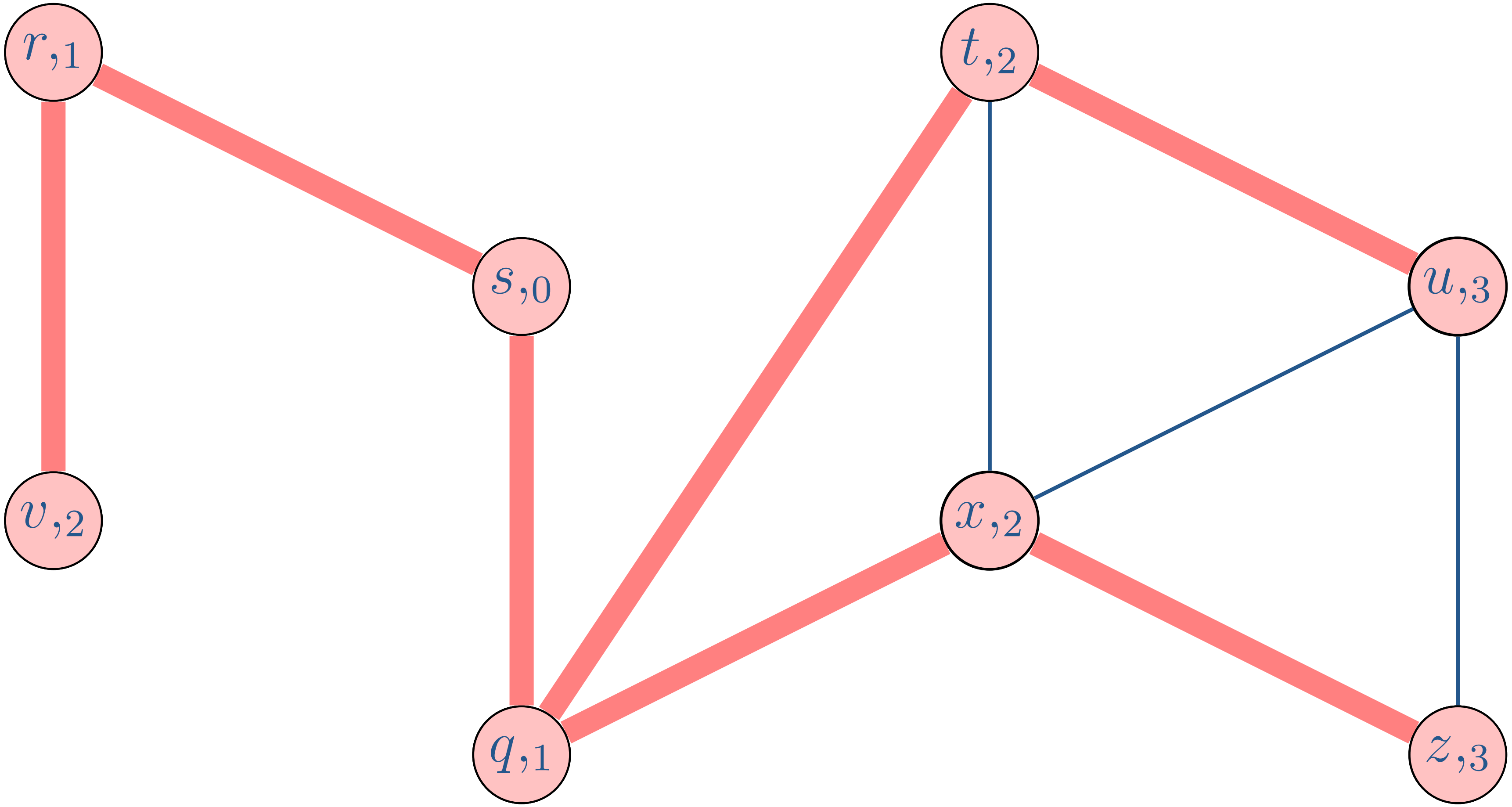
$Q = uz$

BFS – příklad



$Q = z$

BFS – příklad



$Q = \emptyset$

Jaká je složitost BFS?

```
BFS(G, s)
color[s] ← GREY
d[s] ← 0
π[s] ← NIL
for každý vrchol u ∈ V − {s}
    do color[u] ← WHITE
       d[u] ← ∞
       π[u] ← NIL
InitQueue(Q)
Add(Q, s)
while not IsEmpty(Q)
    do u ← Front(Q)
       for každý v ∈ Adj[u]
           do if color[v] = WHITE
                 then color[v] ← GREY
                      d[v] ← d[u] + 1
                      π[v] ← u
                      Add(Q, v)
       Remove(Q)
       color[u] ← BLACK
```

Analýza složitosti BFS

- Ve **while**-cyklu již není možno obarvit vrchol na bílo.
- Řádek 13 proto zaručuje, že každý vrchol bude zařazen do fronty (a následně vybrán z fronty) nejvýše jednou.
- Operace vkládání a vybírání prvku z fronty je konstantní, tj. $O(1)$, takže celkový čas na operace fronty je $O(n)$.
- Protože se seznam sousedů daného vrcholu prochází pouze při jeho vybrání z fronty, je seznam skenován nejvýše jednou.
- Jelikož je suma délek těchto seznamů rovna $\Theta(m)$, je celkový čas skenování seznamu sousedů $O(m)$.
- Inicializace zabere čas $O(n)$, proto je celkový čas algoritmu $O(m + n)$, tj. lineární vzhledem k velikosti reprezentace grafu G.

Prohledávání grafu do hloubky

- Anglicky *Depth-First Search (DFS)*
- DFS odpovídá procházení bludištěm.
- Vstup: (ne)orientovaný graf $G = (V, E)$ a vrchol $s \in V$.
- Výstup: **strom prohledávání do hloubky**
- Reprezentace grafu – seznam sousedů.
- Používá pole předchůdců π.
- Varianta: projít všechny vrcholy a vytvořit **les prohledávání do hloubky**.

Prohledávání grafu do hloubky

- Při procházení grafu obarvuje vrcholy bílou, šedou a černou barvou.

  $color[u] \in \{$WHITE, GREY, BLACK$\}$

- $d[u]$ je časová známka prvního prozkoumání vrcholu (obarvení na šedo).
- $f[u]$ je časová známka dokončení prozkoumávání seznamu sousedů vrcholu u (začernění vrcholu).

  $color[u] =$ *WHITE* v čase před $d[u]$

  $color[u] =$ *GREY* v čase $d[u]$ až $f[u]$

  $color[u] =$ *BLACK* v čase po $f[u]$

- $time$ je globální proměnná.

DFS(G, s)

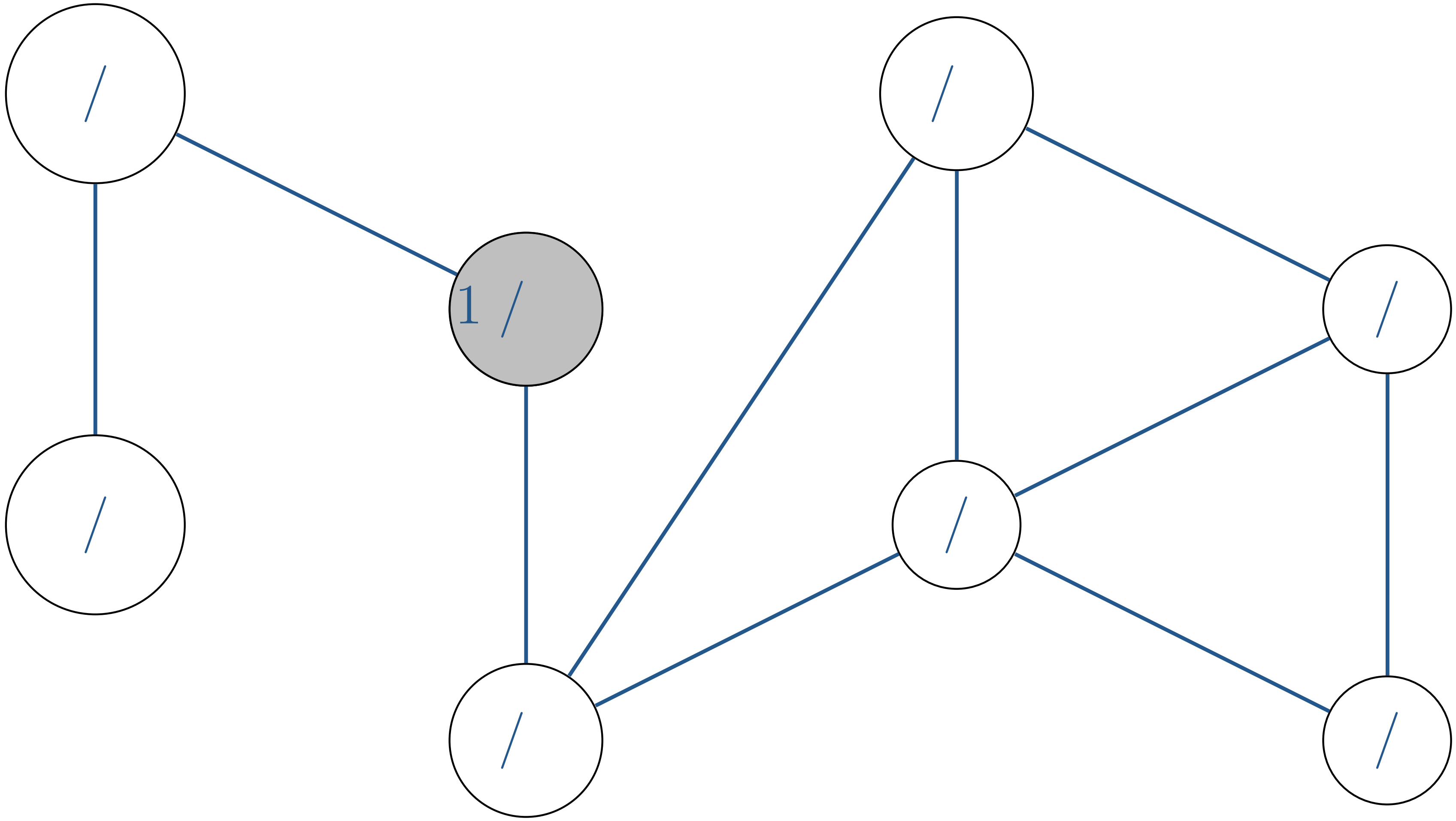
DFS(G, s)

for KAŽDÝ VRCHOL $u \in V$
    do $color[u] \leftarrow$ *WHITE*
        $\pi[u] \leftarrow$ *NIL*
$time \leftarrow 0$
DFS-VISIT(s)

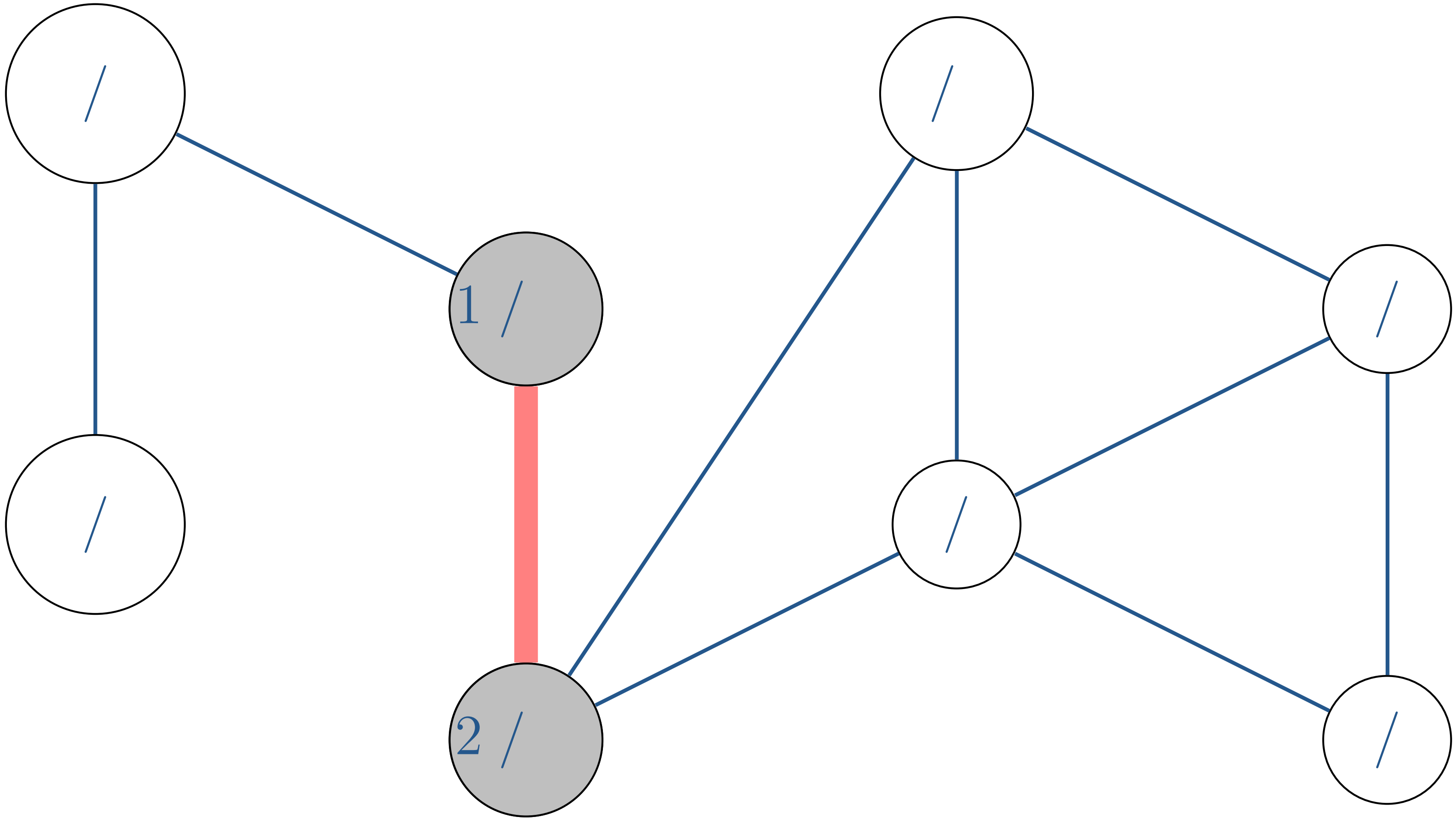
DFS-VISIT(u)

$color[u] \leftarrow$ *GREY*
$time \leftarrow time + 1$
$d[u] \leftarrow time$
for KAŽDÝ VRCHOL $v \in Adj[u]$
    do if $color[v] =$ *WHITE*
        then $\pi[v] \leftarrow u$
            DFS-VISIT(v)
$color[u] \leftarrow$ *BLACK*
$time \leftarrow time + 1$
$f[u] \leftarrow time$

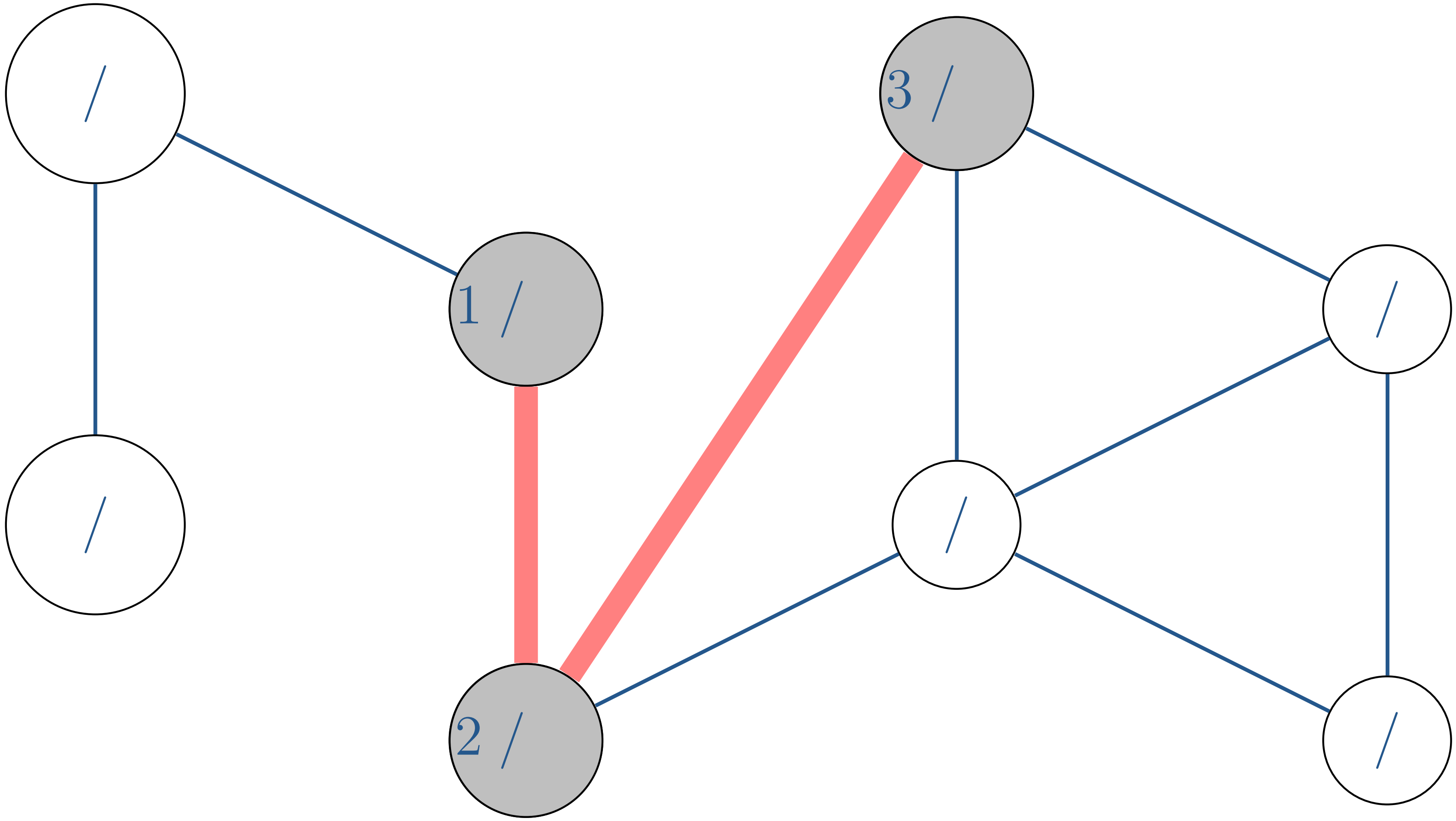
DFS – příklad



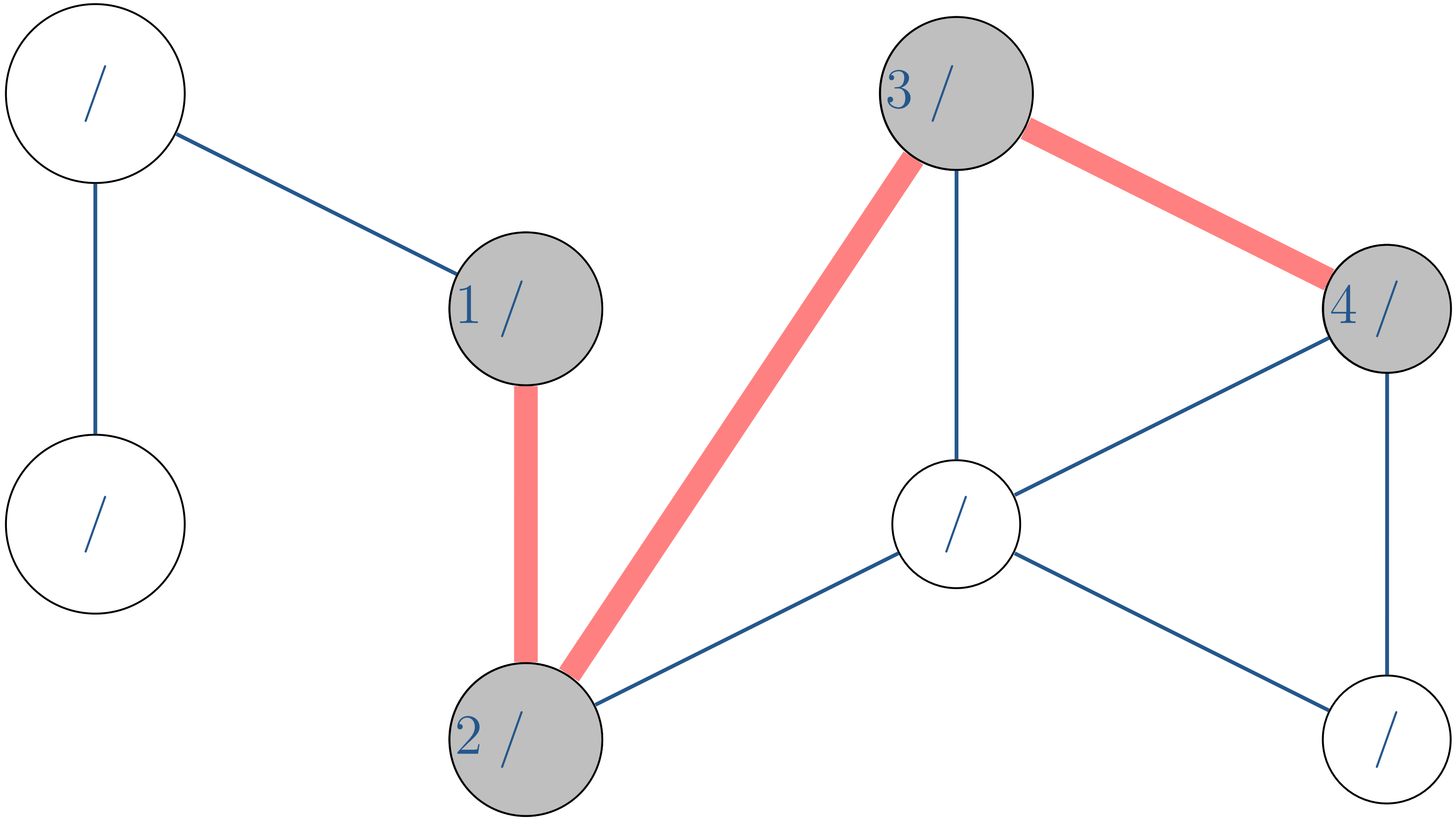
DFS – příklad



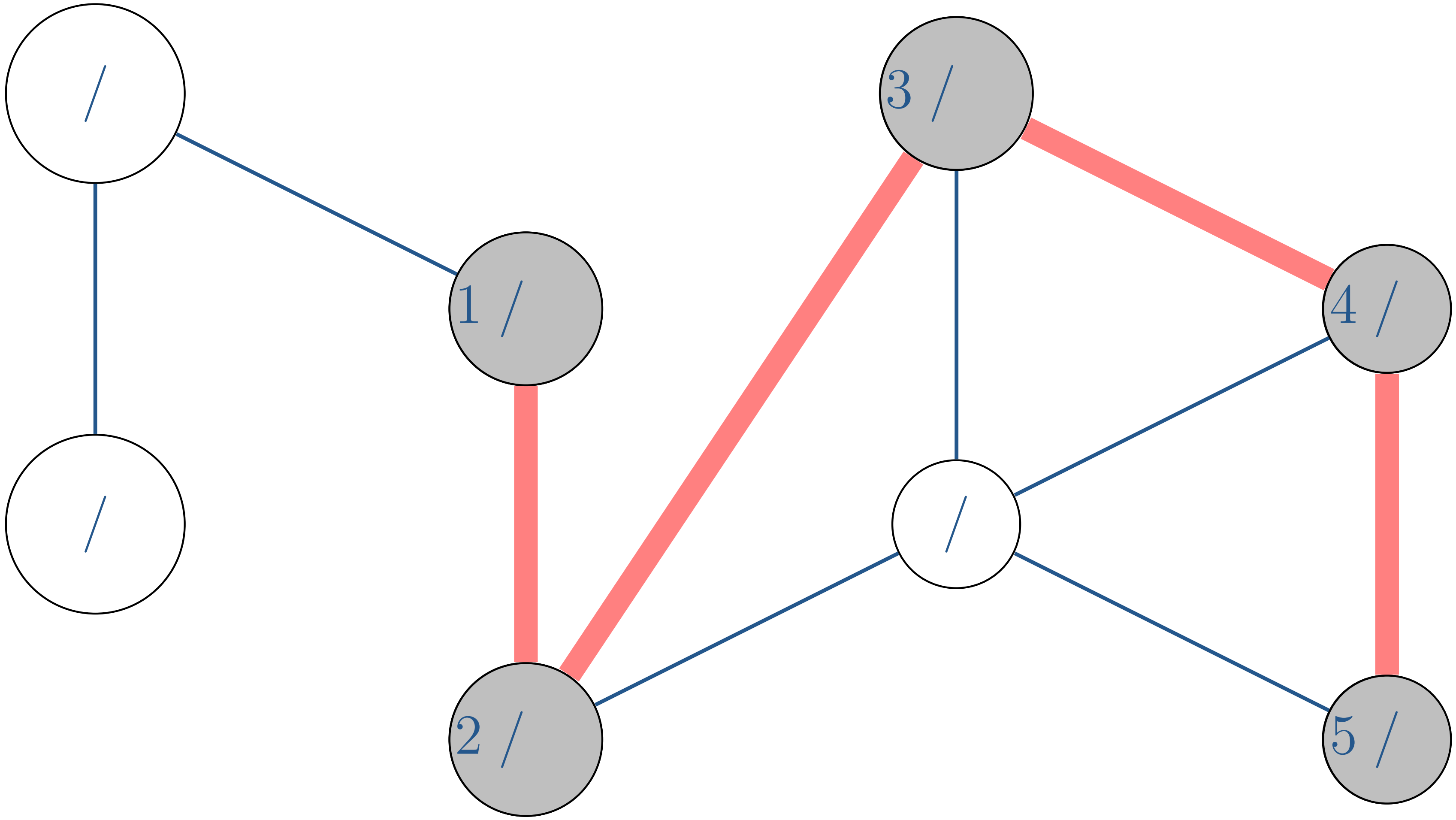
DFS – příklad



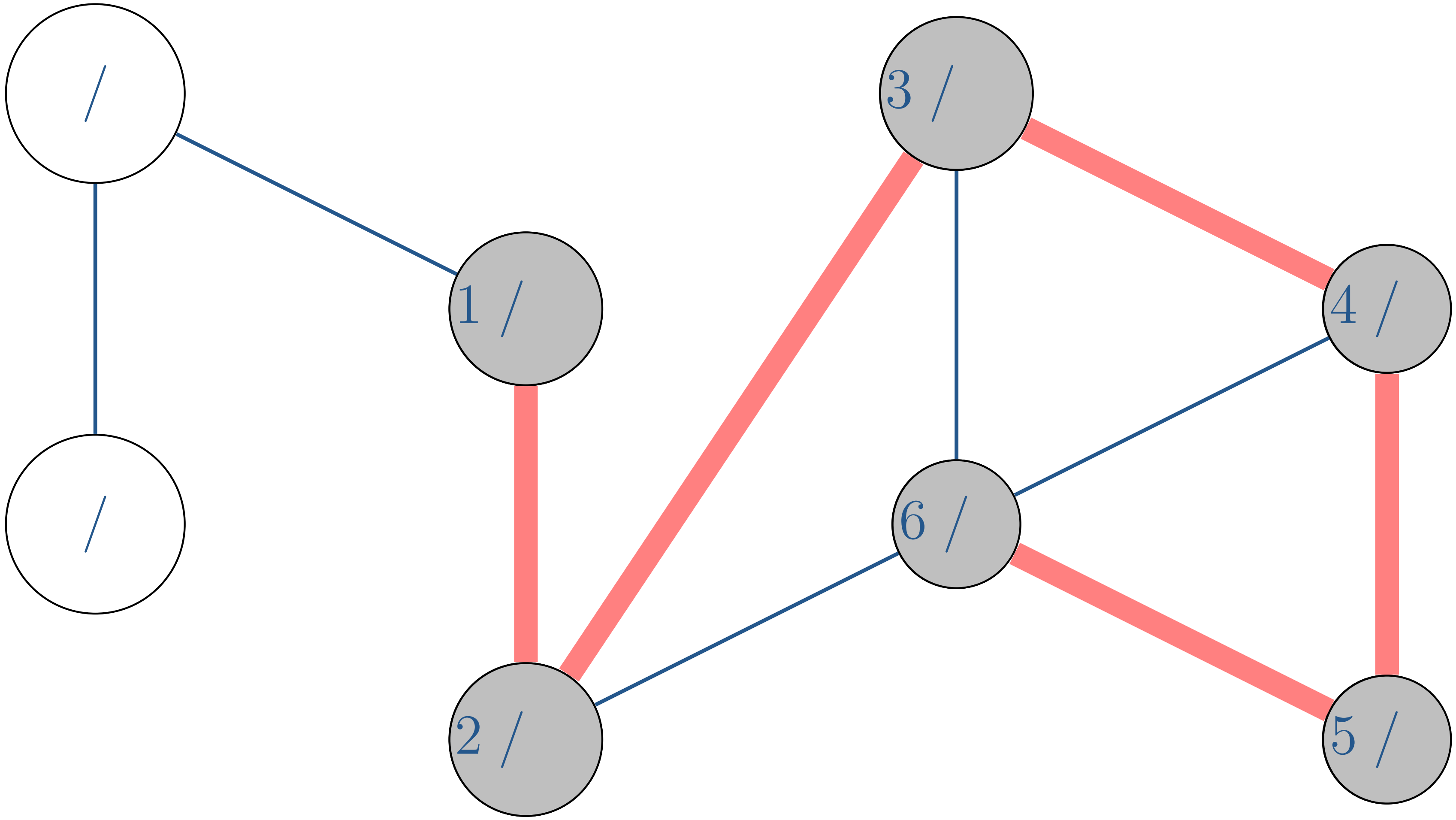
DFS – příklad



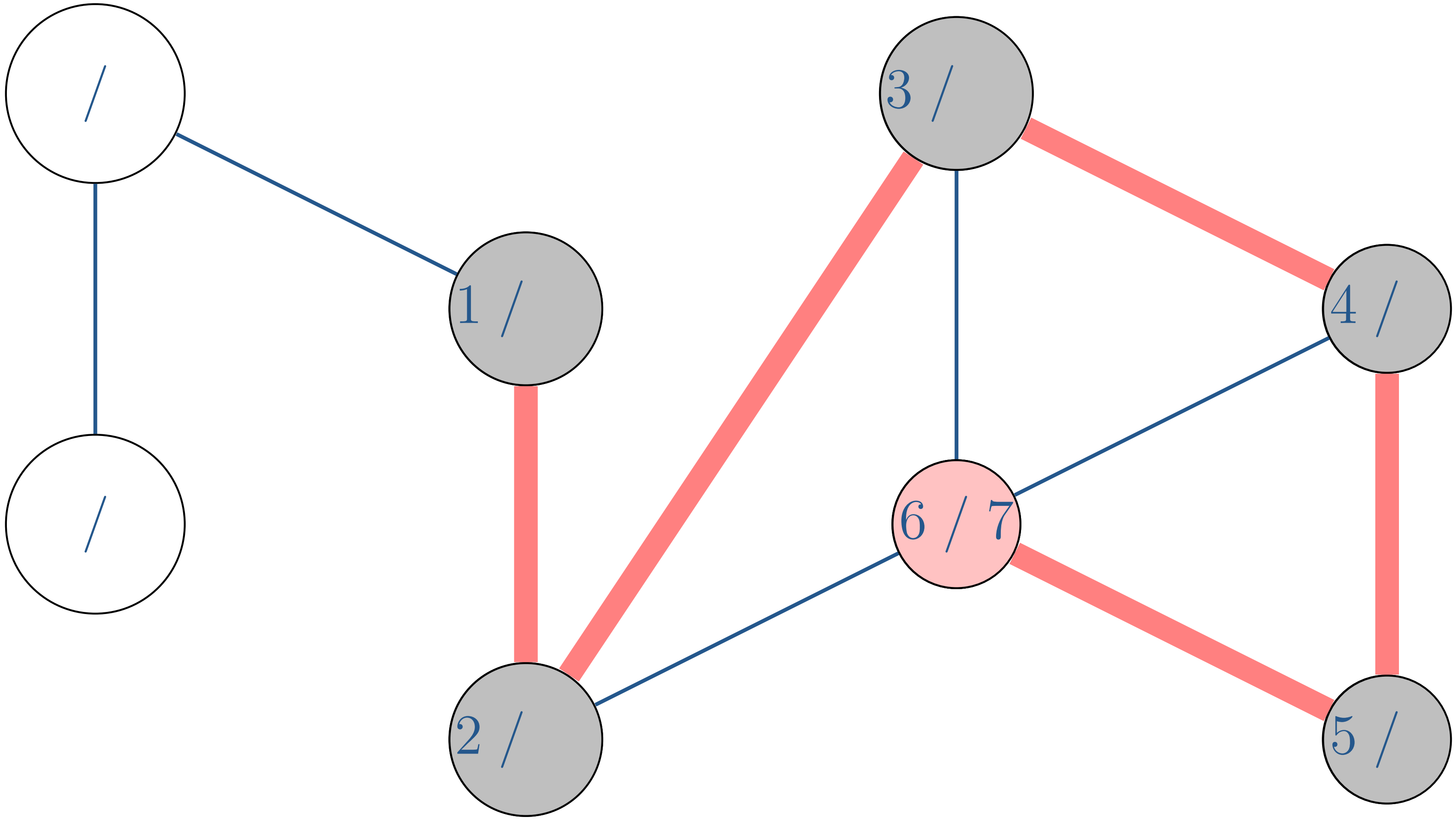
DFS – příklad



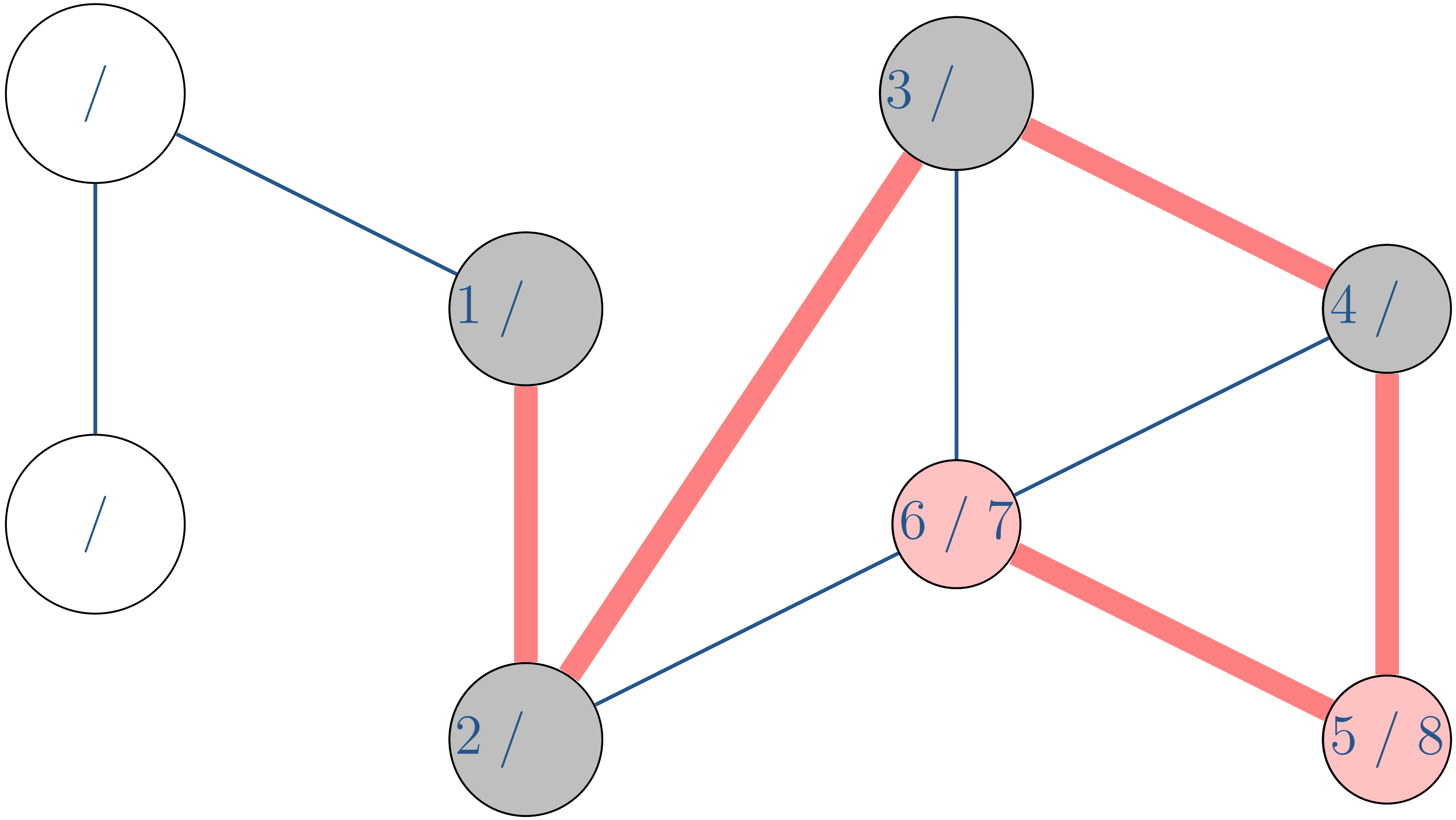
DFS – příklad



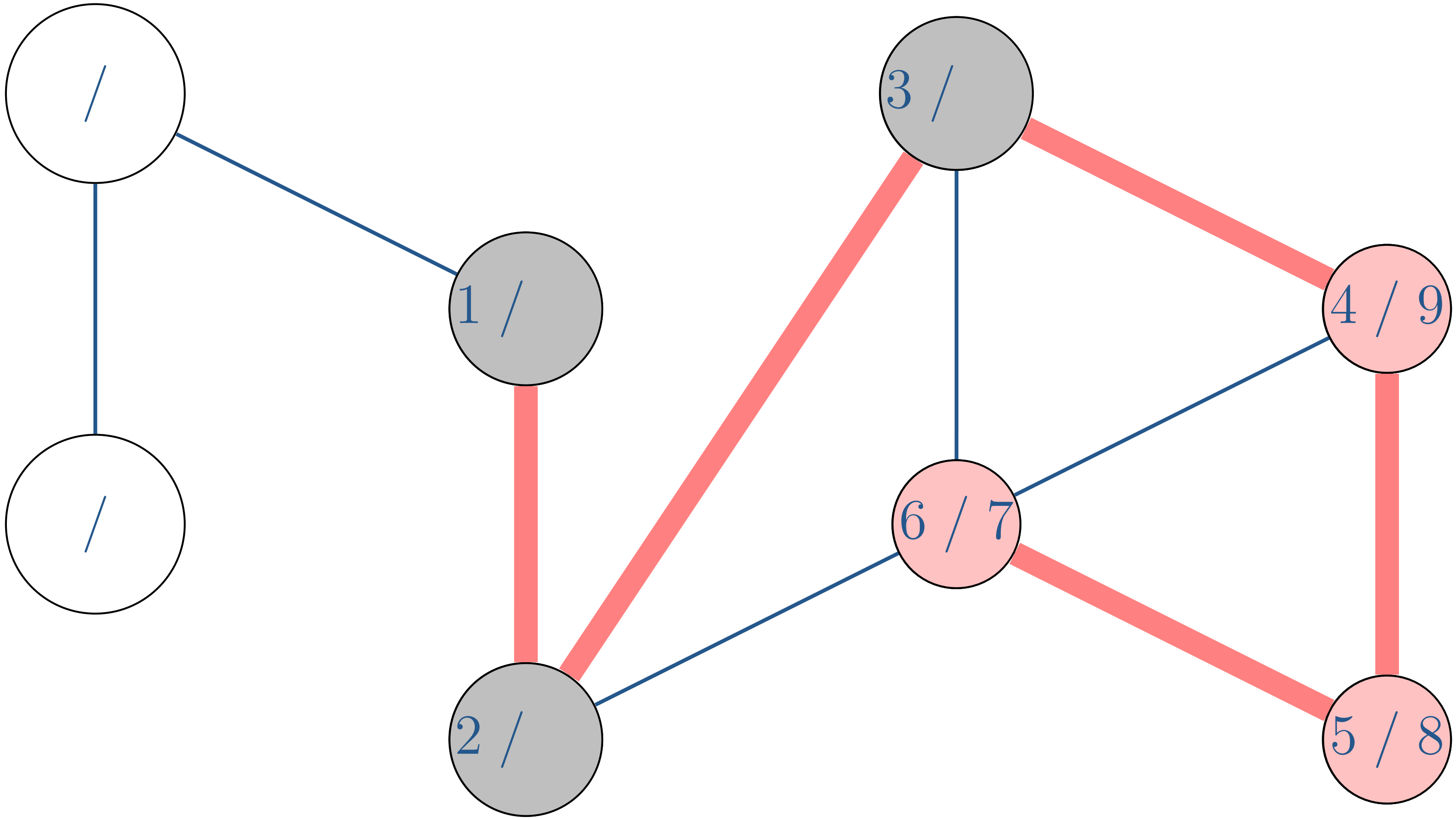
DFS – příklad



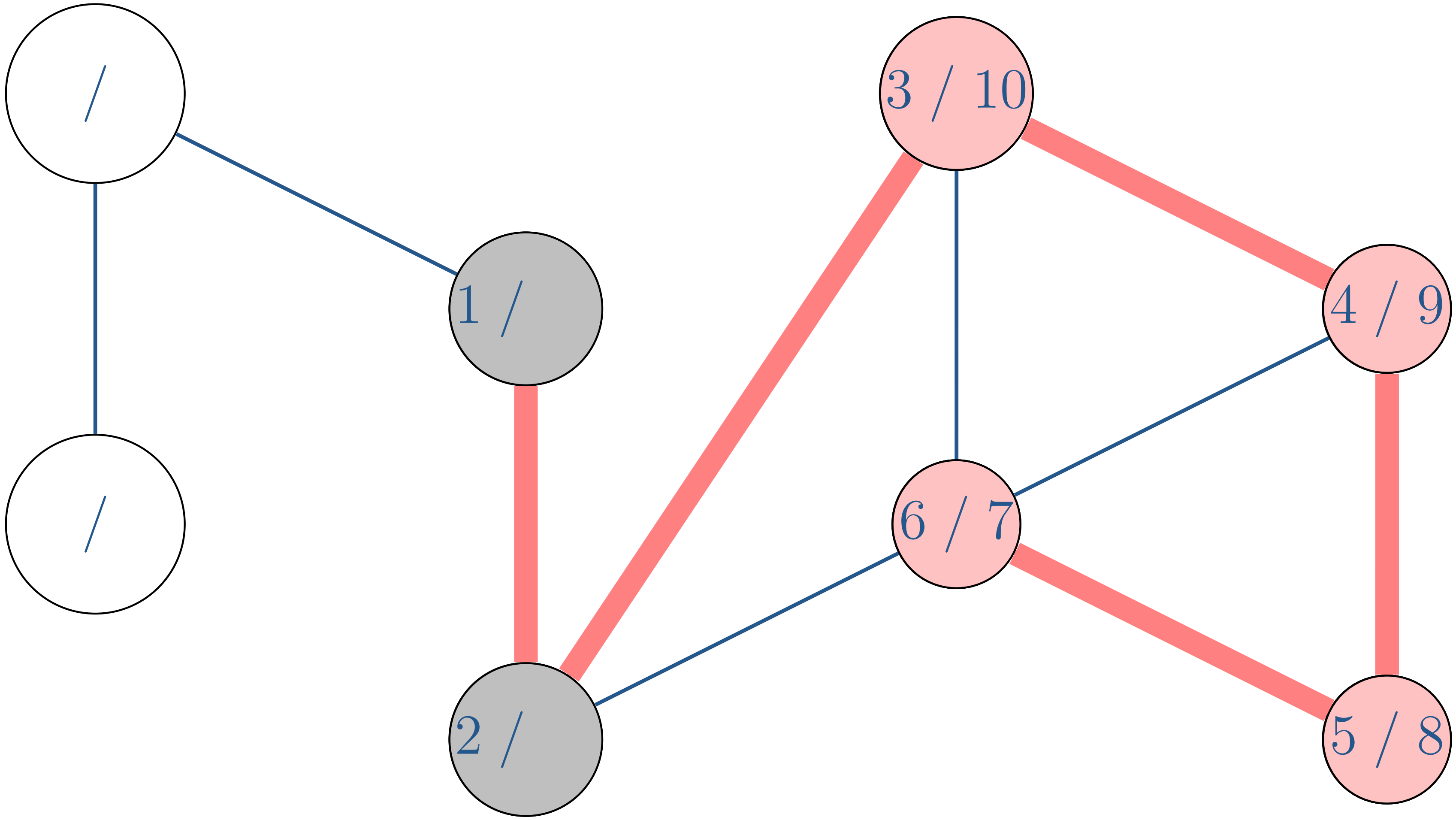
DFS – příklad



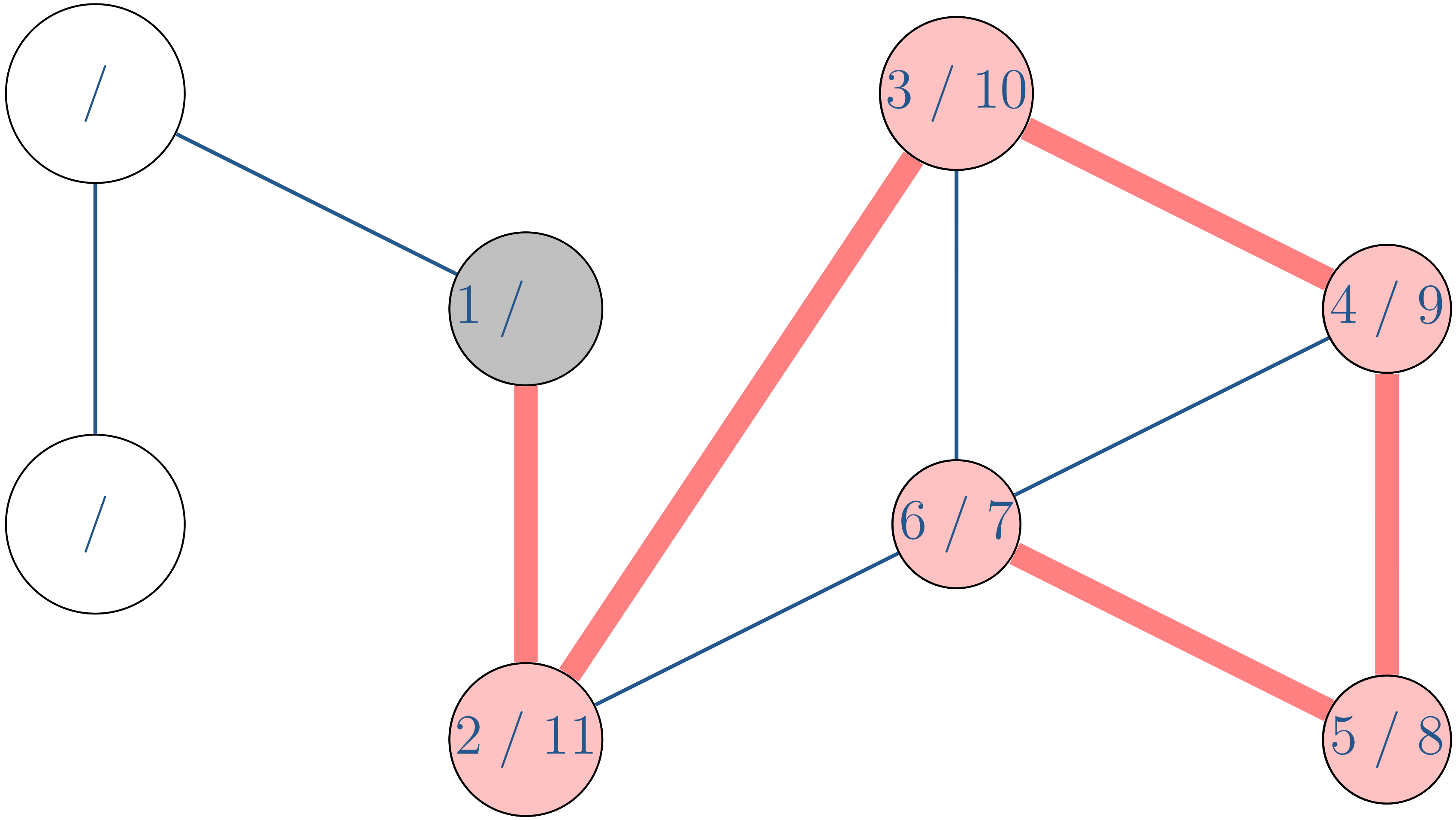
DFS – příklad



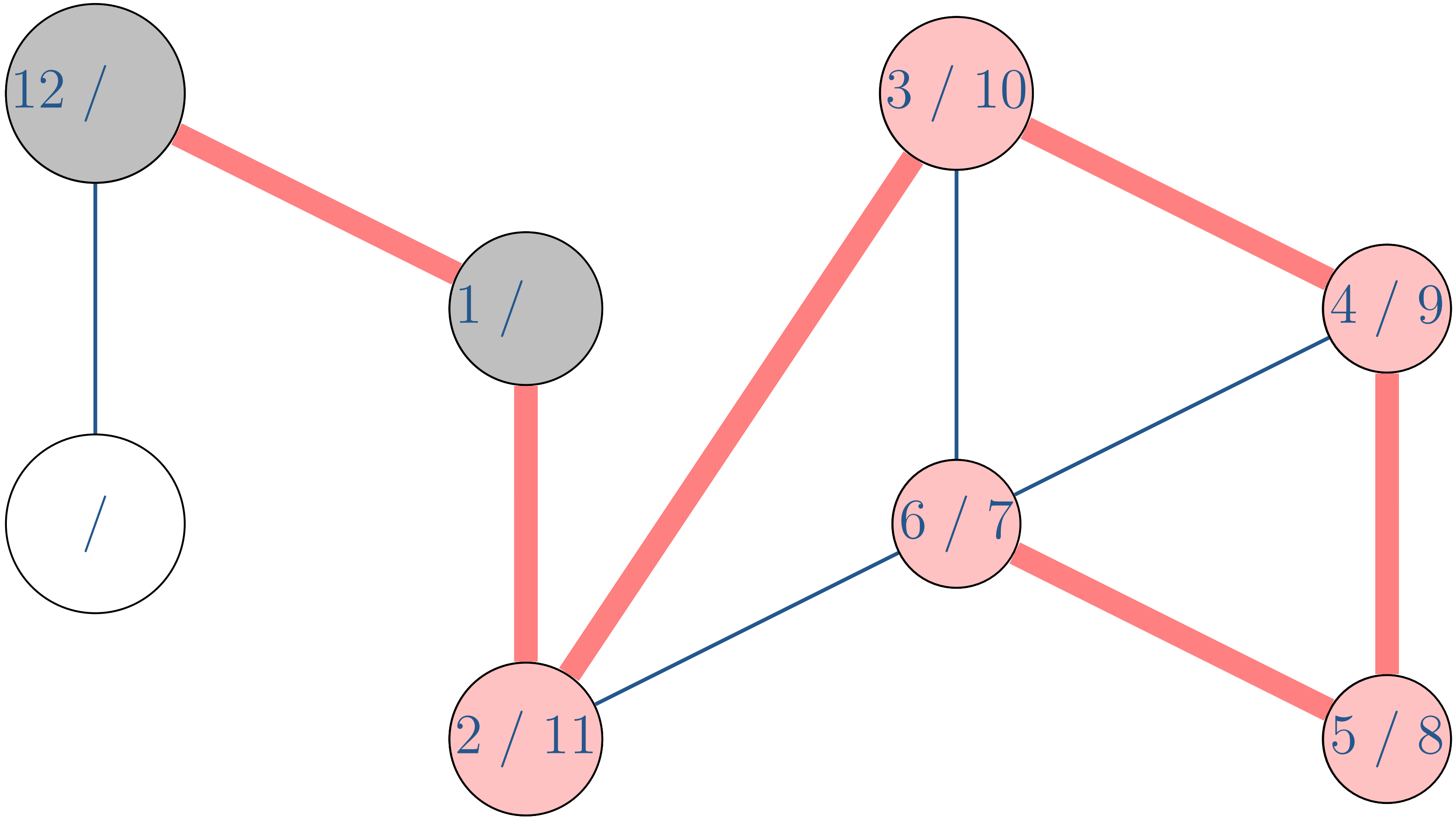
DFS – příklad



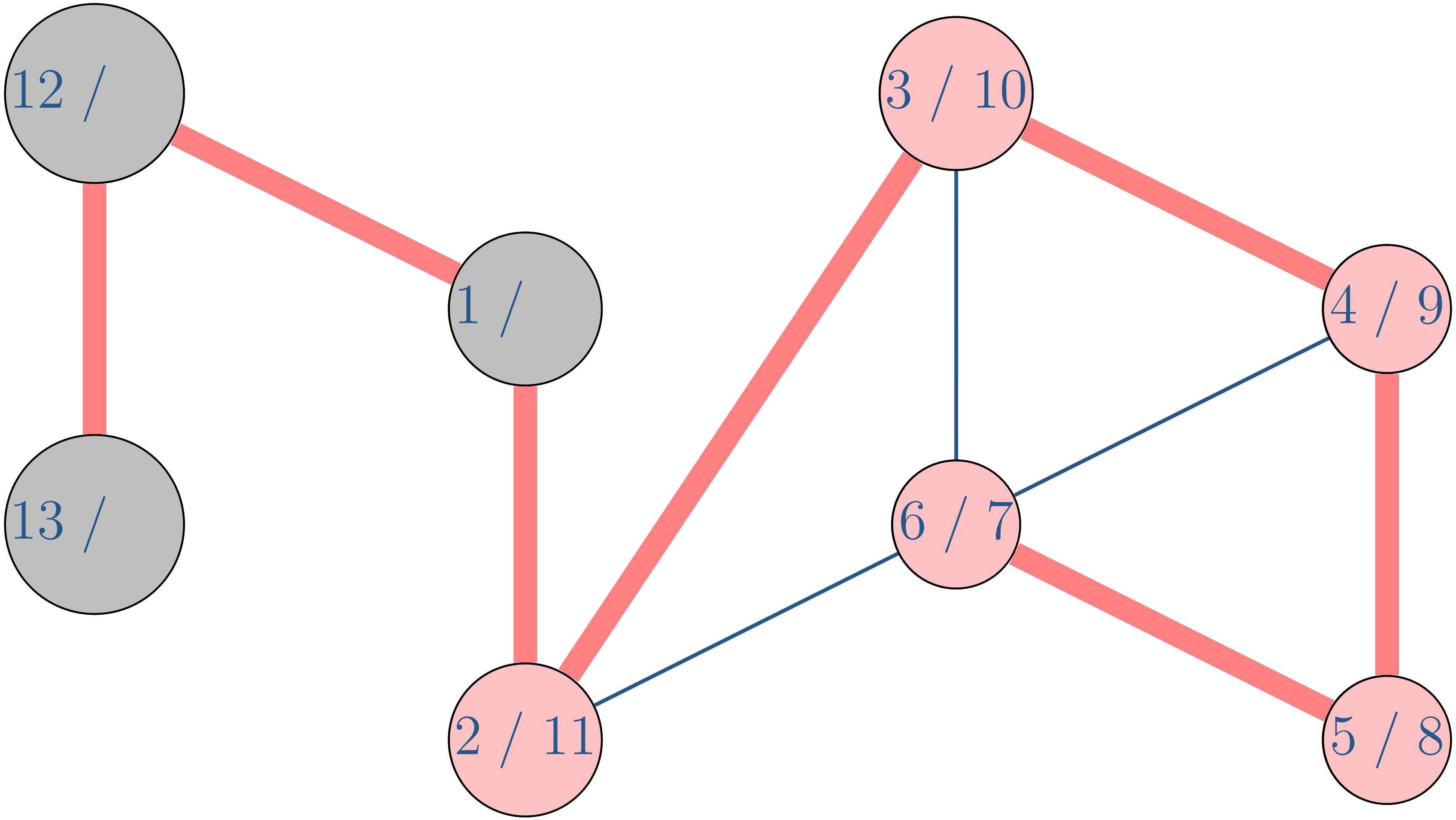
DFS – příklad



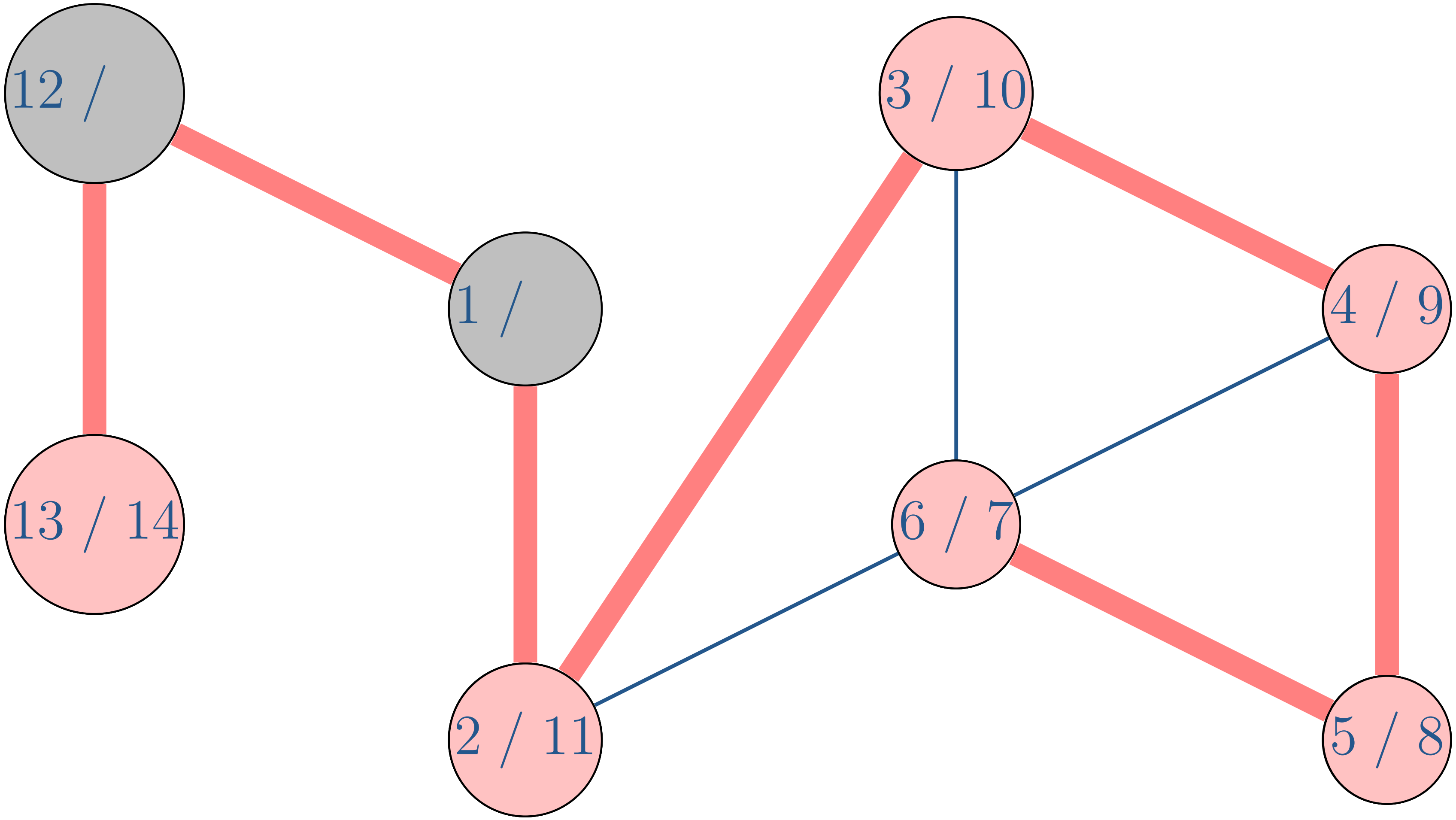
DFS – příklad



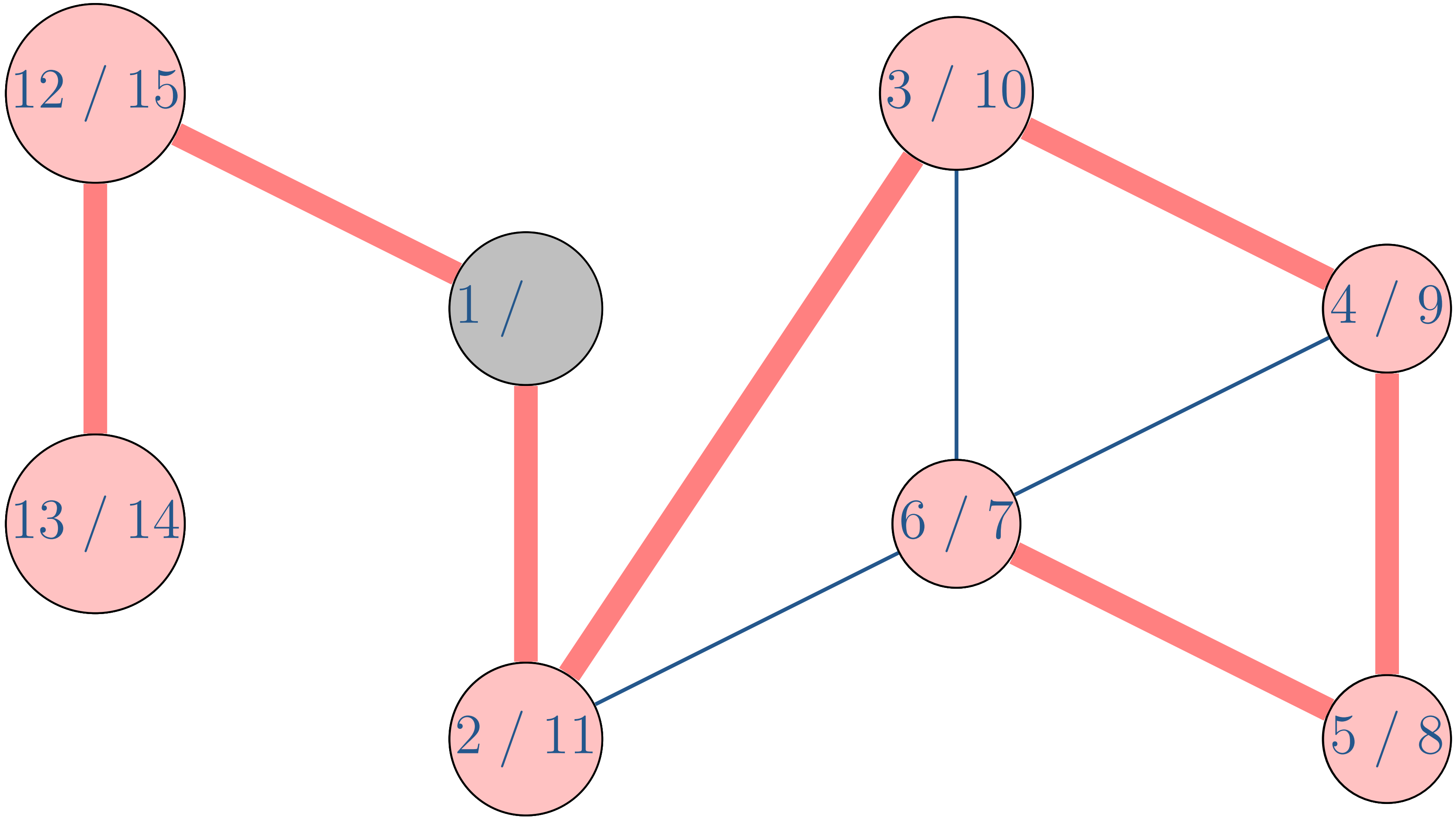
DFS – příklad



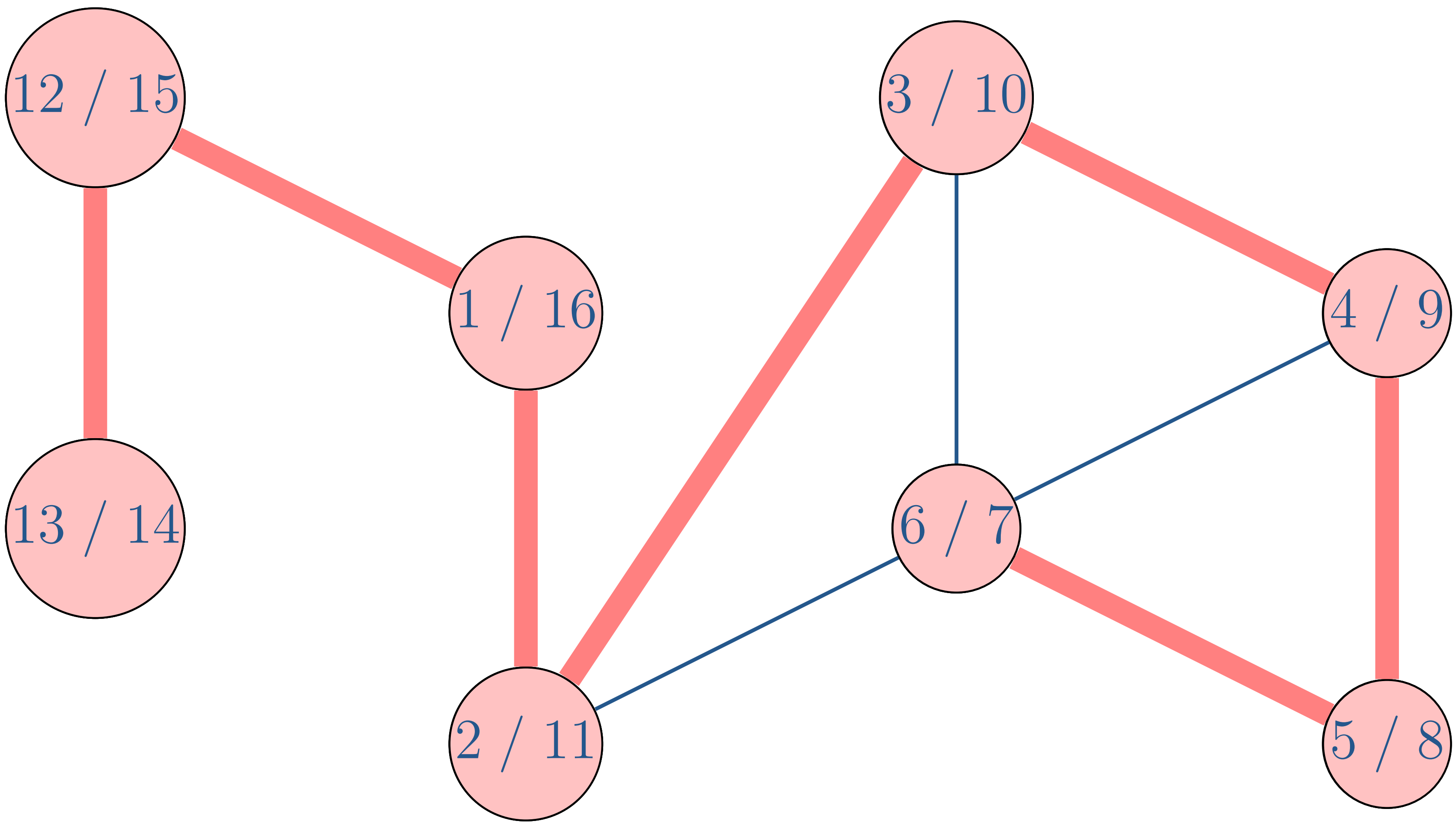
DFS – příklad



DFS – příklad



DFS – příklad



Jaká je složitost DFS?

DFS(G, s)

for KAŽDÝ VRCHOL $u \in V$
    do $color[u] \leftarrow$ *WHITE*
        $\pi[u] \leftarrow$ *NIL*
$time \leftarrow 0$
DFS-VISIT(s)

DFS-VISIT(u)

$color[u] \leftarrow$ *GREY*
$time \leftarrow time + 1$
$d[u] \leftarrow time$
for KAŽDÝ VRCHOL $v \in Adj[u]$
    do if $color[v] =$ *WHITE*
        then $\pi[v] \leftarrow u$
            DFS-VISIT(v)
$color[u] \leftarrow$ *BLACK*
$time \leftarrow time + 1$
$f[u] \leftarrow time$

Analýza složitosti DFS

- Inicializace na řádcích 1–3 zabere čas $\Theta(n)$.
- Funkce DFS-VISIT(v) je volána pouze pro bílé vrcholy a první věc, kterou udělá, je jejich obarvení na šedo. Je tedy volána nejvýše jednou pro každý vrchol $v \in V$.
- Pro každý vrchol v je cyklus 9–12 prováděn $|Adj[v]|$-krát.
- Jelikož $\sum_{v \in V} |Adj[v]| = \Theta(m)$, je celková cena řádků 9–12 $O(m)$.
- Není-li totiž G souvislý, do některých vrcholů se vůbec nedostaneme.
- Celková složitost je tedy $O(m + n)$.

Nejkratší cesty z jednoho do všech vrcholů

- Máme ohodnocený orientovaný graf $G = (V, E)$
  s váhovou funkcí $w : E \to \mathbb{R}$.
- **Cena cesty** $p = \langle v_0, v_1, \ldots, v_k \rangle$ je suma $w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$.
- **Cena nejkratší cesty** z u do v je
  - $\delta(u, v)$ – minimum $w(p)$, pokud existuje cesta p z u do v,
  - ∞ jinak.
- **Nejkratší cesta** z u do v je pak libovolná cesta p z u do v
  s minimální cenou $\delta(u, v)$.

- Varianty nejkratší cesty
  - **Ze všech vrcholů do jednoho** – převrátíme orientaci hran
  - **Z jednoho do jednoho**
  - **Ze všech do všech**
  - Graf se zápornými hranami (bez dosažitelného záporného cyklu)

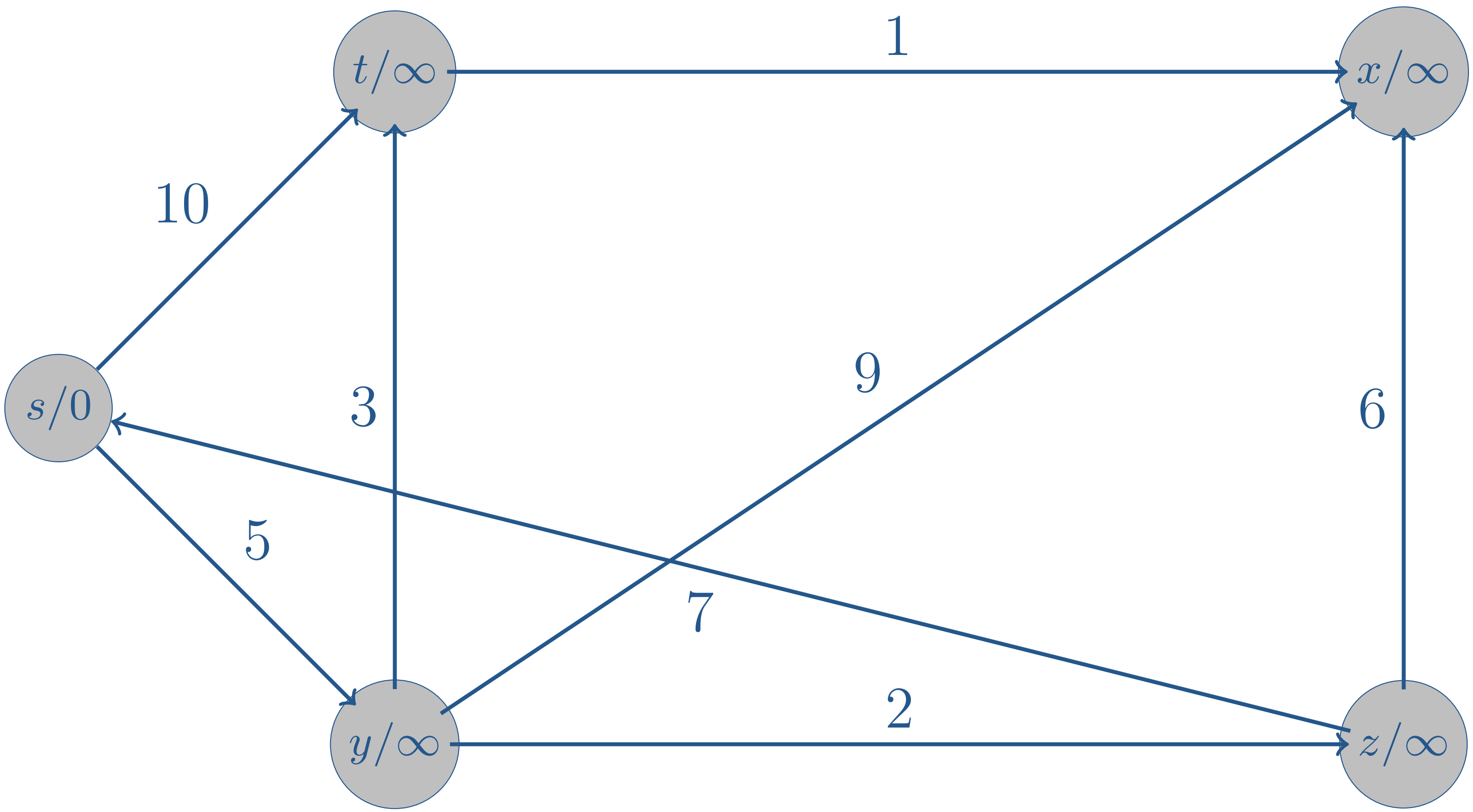
Dijkstrův algoritmus

$d[v]$ – odhad nejkratší cesty
S – vrcholy s vypočtenou nejkratší vzdáleností od s
Q – prioritní fronta (na začátku prvek s min. d-hodnotou)

DIJKSTRA(G, w, s)

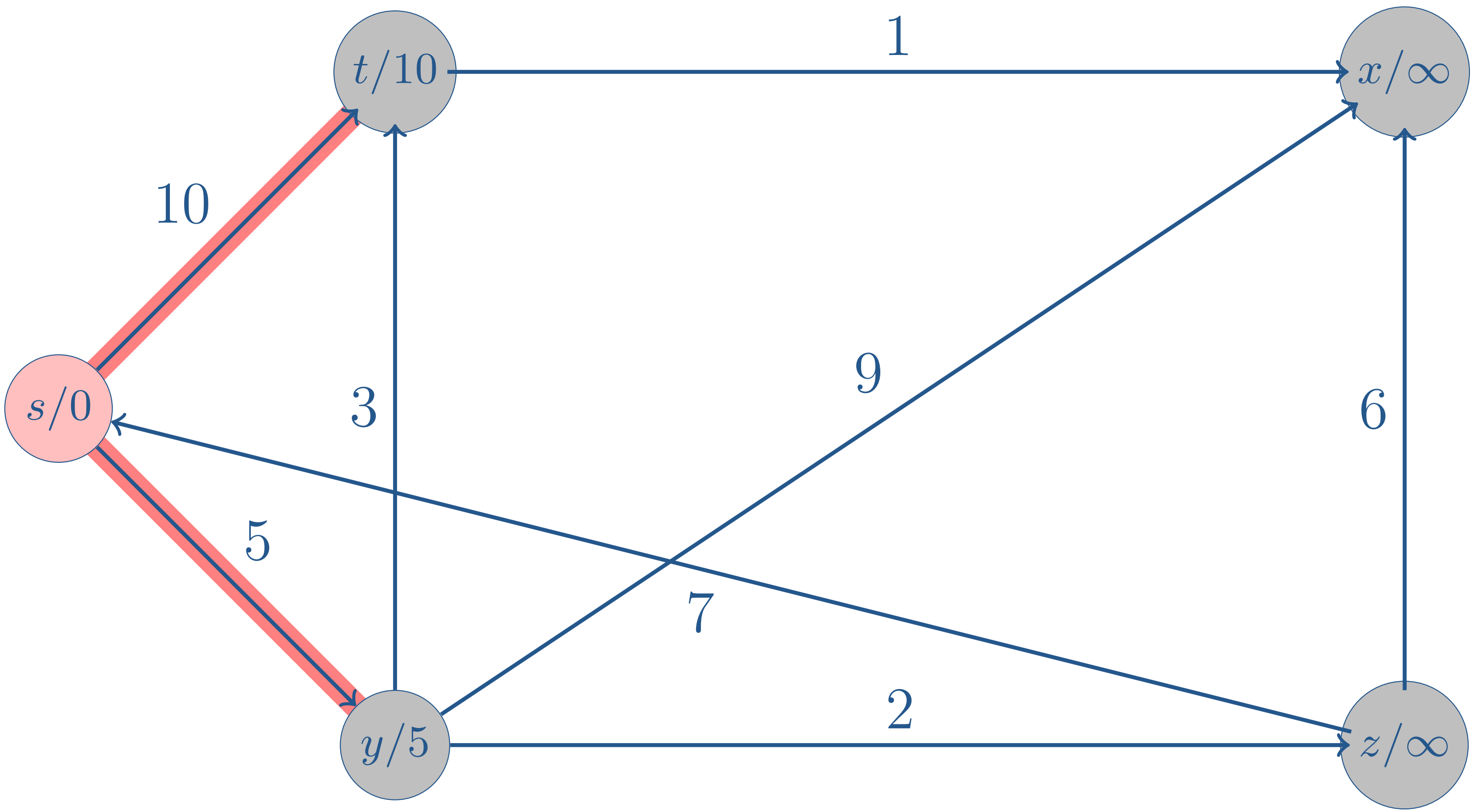
for KAŽDÝ VRCHOL $v \in V$
    do $d[v] \leftarrow \infty$
        $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for KAŽDÝ VRCHOL $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$
                    $\pi[v] \leftarrow u$

Dijkstrův algoritmus – příklad



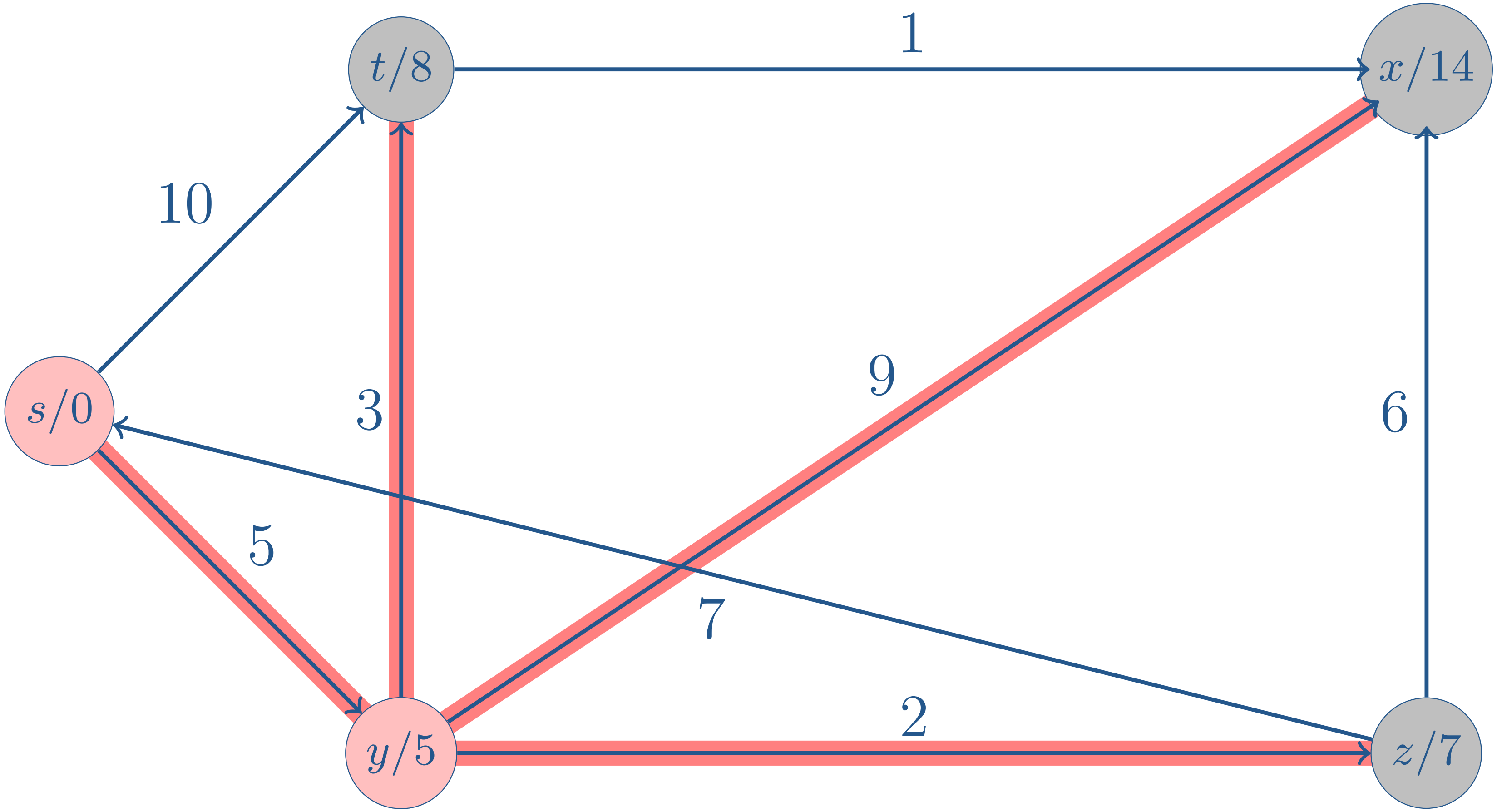
Označené vrcholy značí vrcholy z množiny S (min. vzdálenost spočítána).

Dijkstrův algoritmus – příklad



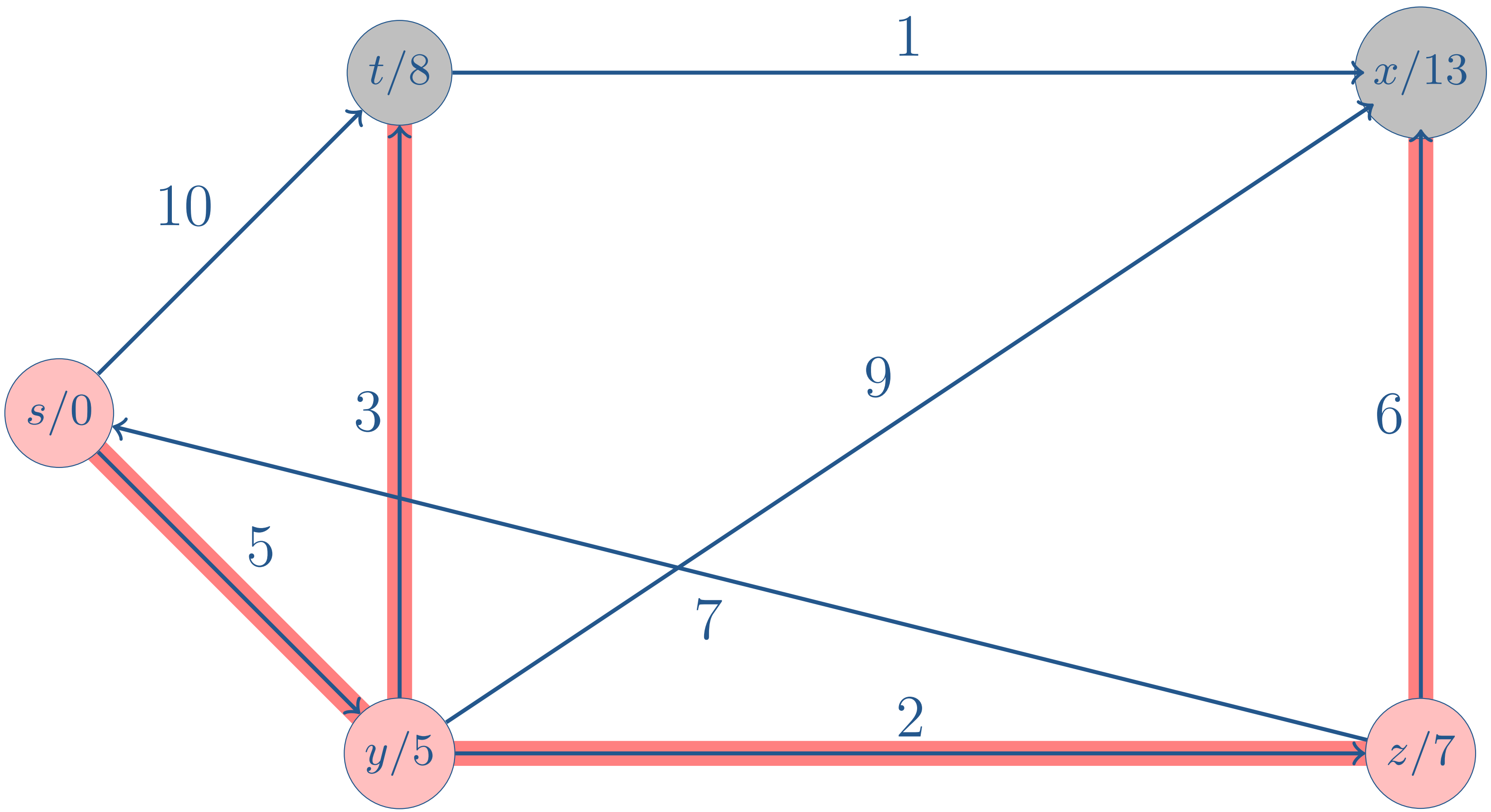
Označené vrcholy značí vrcholy z množiny S (min. vzdálenost spočítána).

Dijkstrův algoritmus – příklad



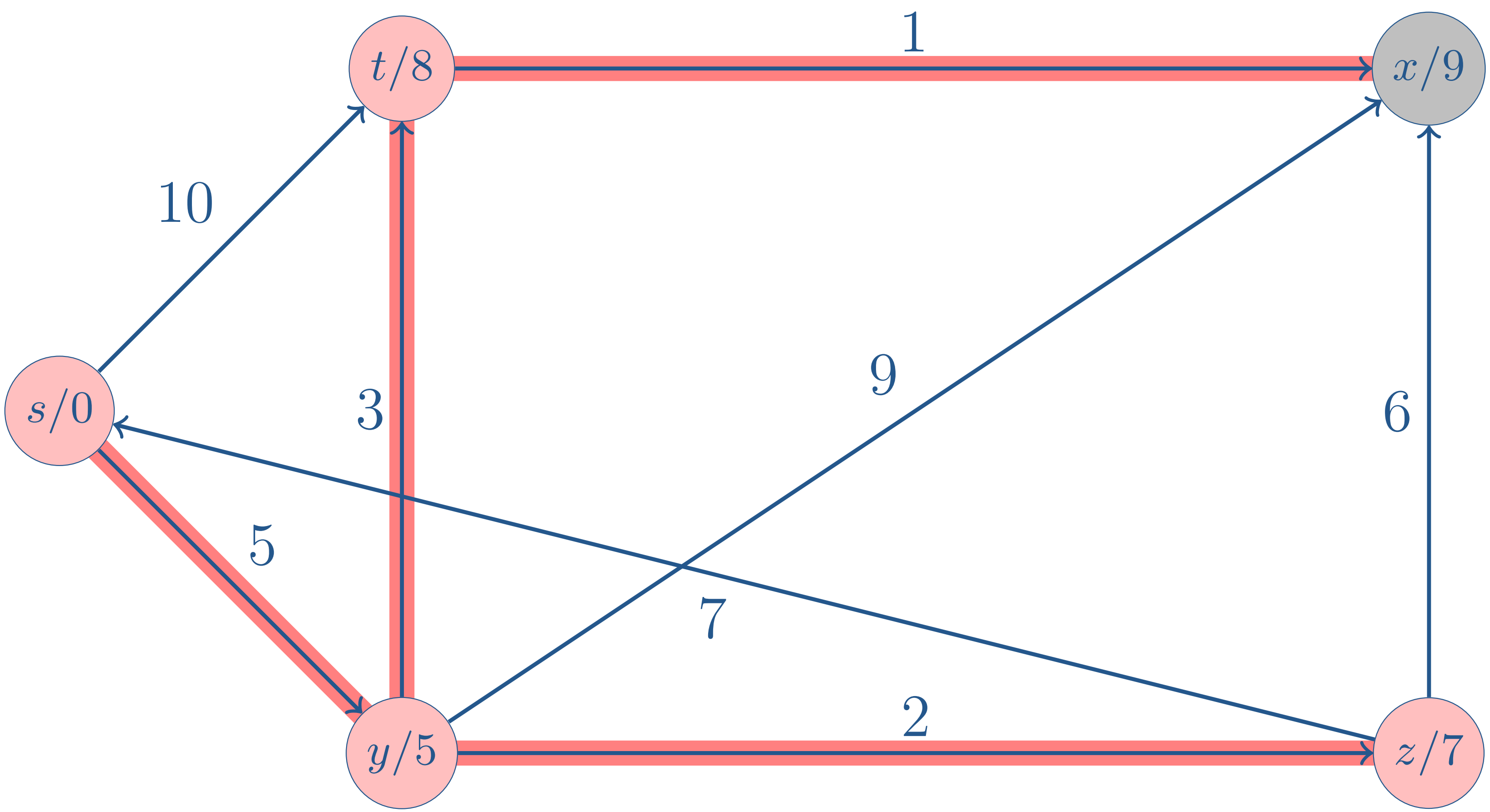
Označené vrcholy značí vrcholy z množiny S (min. vzdálenost spočítána).

Dijkstrův algoritmus – příklad



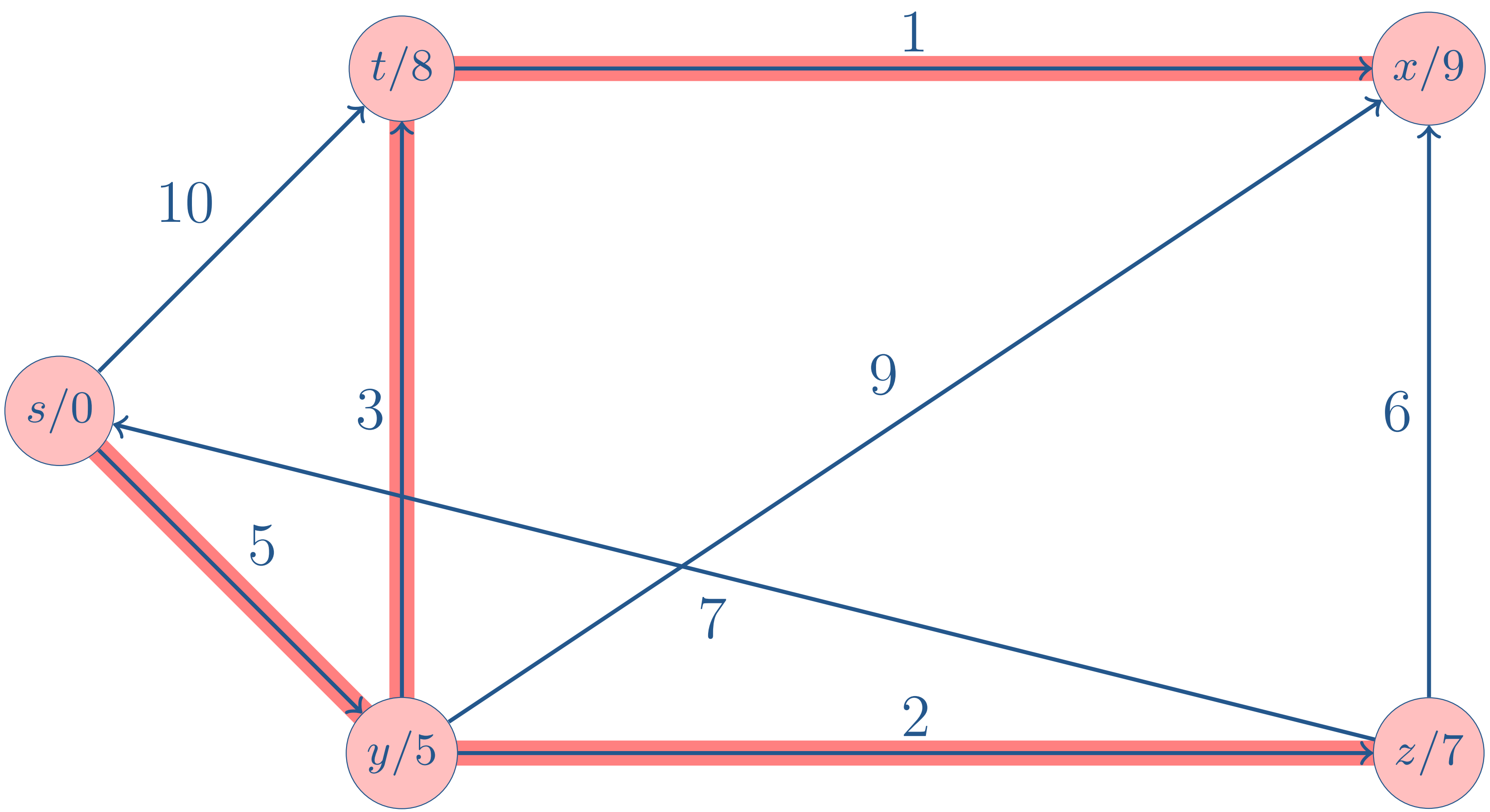
Označené vrcholy značí vrcholy z množiny S (min. vzdálenost spočítána).

Dijkstrův algoritmus – příklad



Označené vrcholy značí vrcholy z množiny S (min. vzdálenost spočítána).

Dijkstrův algoritmus – příklad



Označené vrcholy značí vrcholy z množiny S (min. vzdálenost spočítána).

Jaká je složitost Dijkstrova algoritmu?

$d[v]$ – odhad nejkratší cesty
S – vrcholy s vypočtenou nejkratší vzdáleností od s
Q – prioritní fronta (na začátku prvek s min. d-hodnotou)

DIJKSTRA(G, w, s)

for KAŽDÝ VRCHOL $v \in V$
    do $d[v] \leftarrow \infty$
        $\pi[v] \leftarrow$ NIL
$d[s] \leftarrow 0$
$S \leftarrow \emptyset$
$Q \leftarrow V$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN(Q)
        $S \leftarrow S \cup \{u\}$
        for KAŽDÝ VRCHOL $v \in Adj[u]$
            do if $d[v] > d[u] + w(u, v)$
                then $d[v] \leftarrow d[u] + w(u, v)$
                    $\pi[v] \leftarrow u$

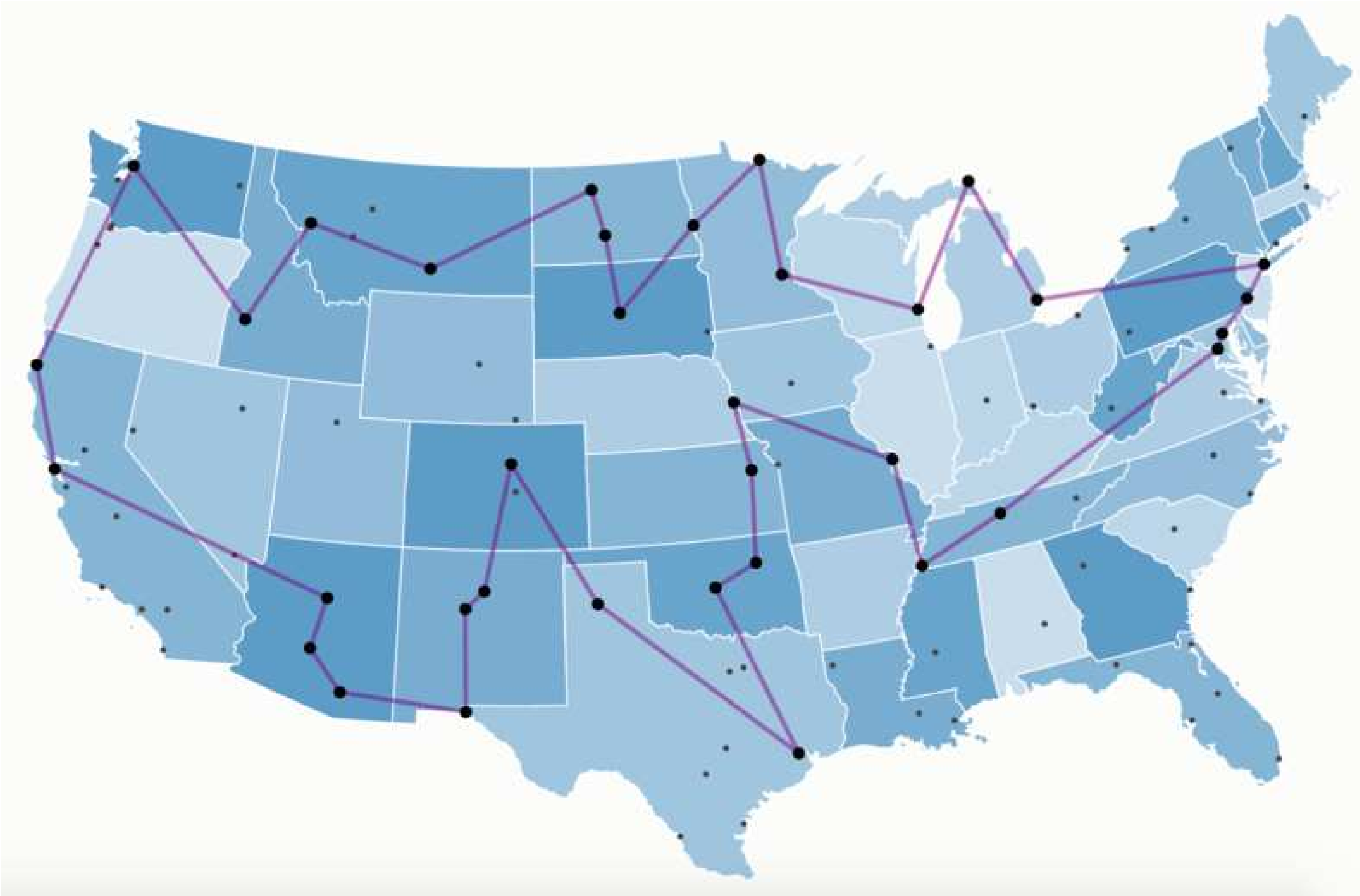
Analýza složitosti Dijkstrova algoritmu

- Inicializace (řádky 1 až 6) proběhne pro každý vrchol, tedy $O(n)$.
- Výběr minimálního prvku z fronty (řádek 8) se provede pro každý vrchol jednou, přičemž při implementantaci prioritní fronty pomocí pole to bude v čase $O(n)$.
- Kontrola a případná aktualizace vzdálenosti $d[v]$ na řádcích 10 až 13 se provede pro každou hranu, tedy m-krát.
- Celkem tedy dostaneme $O(n^2 + m) = O(n^2)$.
- Při implementaci prioritní fronty pomocí binární či Fibonnaciho haldy lze složitost snížit na $O(n \log n + m)$.

Problém obchodního cestujícího

- Anglicky *Travelling Salesman Problem (TSP)*
- Máme n měst, která jsou spojená cestami o známých délkách.
- Úkolem je najít nejkratší trasu, která prochází všemi městy (každým právě jednou) a která se vrací do výchozího města.
- Z pohledu teorie grafů to znamená najít v ohodnoceném úplném grafu nejkratší Hamiltonovskou kružnici (prochází právě jednou všemi vrcholy).
- Ohodnocení hran může odpovídat
  - vzdálenostem mezi městy (nejkratší trasa),
  - času pro přesun z jednoho města do druhého (nejrychlejší trasa),
  - ceně cesty mezi městy (nejlevnější trasa).

Problém obchodního cestujícího – ilustrace



Řešení TSP hrubou silou (Brute-force)

- Vytvoříme seznam všech možných Hamiltonovských kružnic:
  - Začátek (a konec) je dán.
  - V prvním kroku máme na výběr z $|V| - 1$ měst.
  - Ve druhém z $|V| - 2$ měst.
  - $\ldots$
- Sečteme ohodnocení jejich hran.
- Vybereme kružnici s nejnižším ohodnocením.

Analýza složitosti

- Celkem existuje $(|V| - 1)!$ kružnic.
- Každá kružnice má $|V|$ hran.
- Je tedy třeba zpracovat $|V|!$ hran – $O(n!)$.

Je to hodně nebo málo?

Řešení hrubou silou – náročnost výpočtu

Předpokládejme rychlost zpracování 1 000 000 000 hran za sekundu.

Vrcholů	Hran ke zpracování	Čas výpočtu
5	120	120 ns
10	3 628 800	3,6 ms
15	1,3 * 10 exp 12	22 minut
20	2,4 * 10 exp 18	77 let
25	1,6 * 10 exp 25	492 milionů let

Zhodnocení

- Hrubou silou se dá TSP rozumně řešit maximálně pro 17 měst (4 dny).

Held-Karpův algoritmus

- Využívá dynamické programování s rozdělením problému na podproblémy a uchováváním mezivýsledků.
- Jeho časová složitost je však stále velká $O(n^2 * 2^n)$.
- Přičemž značná je i jeho paměťová složitost $O(n * 2^n)$.

Vrcholů	Čas výpočtu	Potřebná paměť
10	0,1 ms	10 kB
20	0,4 s	20 MB
30	16,1 s	30 GB
40	20,4 dne	40 TB
50	89 roků	50 PB

- S tímto algoritmem jsme TSP schopni rozumně řešit až pro 38 měst (4,5 dne a 9,5 TB paměti).
- Další známé algoritmy již složitost řešení TSP významněji nezlepšují.

TSP je teoreticky zajímavý problém

- Jedná se o **NP-úplný** (angl. *NP-complete*) problém.
- NP znamená řešitelný nedeterministicky v polynomiálním čase.
- Libovolný NP problém lze v polynomiálním čase převést na NP úplný.
- Další NP-úplné problémy:
  - SAT – splnitelnost logických formulí v CNF
  - Klika – existuje úplný podgraf s k vrcholy?
  - Problém batohu – maximalizace hodnoty věcí v batohu při respektování jeho nosnosti
  - Problém dvou loupežníků – lze skupinu čísel rozdělit na dvě podskupiny tak, aby jejich součet byl stejný?
  - ...
- Více v předmětech Teoretická informatika (TIN) a Složitost (SLO).

Praktické vypořádání se (nejen) s TSP

Jak se tedy vypořádat s problémy jako je TSP?

Praktické vypořádání se (nejen) s TSP

Jak se tedy vypořádat s problémy jako je TSP?

Rezignujeme na nalezení zaručeně nejlepšího řešení a snažíme se v daném čase a s danou pamětí najít co nejlepší řešení – **optimalizační problém**.

Praktické vypořádání se (nejen) s TSP

- Heuristické algoritmy
  - Nejbližší soused (*Nearest Neighbor*) – vybírá nejbližší město
  - Nejbližší vložení (*Closest Insertion*) – na obou koncích dočasné cesty
  - Geometrický algoritmus (*Geometric Algorithm*) – do konvexní obálky postupně přidává vnitřní města, a to s co nejnižší cenou (největším úhlem)
  - . . .
- Pravděpodobnostní algoritmy
  - Metoda MonteCarlo
  - Simulované žíhání (*Simulated Annealing*)
- Genetické algoritmy – inspirace přirozeným výběrem
- Mravenčí kolonie (Ant Colony) – inspirace mravenci a feromony
- Více v předmětech Základy umělé inteligence (IZU), Aplikované evoluční algoritmy (EVO) a Biologií inspirované počítače (BIN).

Paralelní algoritmy

- Sekvenční algoritmy mají své limity.
- Dnes jsou však běžně dostupné paralelní výpočetní systémy:
  - Vícejádrové procesory (multi-core)
  - Víceprocesorové paralelní systémy se sdílenou pamětí
  - Distribuované výpočetní systémy
  - Grafické karty (GPU)
  - Programovatelná hradlová pole (FPGA)
  - $\ldots$
- Máme-li dost procesorů, můžeme snížit i třídu časové složitosti.
- Například paralelní Bubble sort s $n/2$ procesory má složitost $O(n)$.
- Ne vždy je však paralelizace snadná (synchronizace).
- Více v předmětu Paralelní a distribuované algoritmy (PRL).

Souběžný přístup k datovým strukturám (1/2)

- Vezměme si třeba dvojsměrně vázaný lineární seznam a souběžné provedení operací DLL_INSERTAFTER a DLL_DELETEAFTER různými vlákny.

Souběžný přístup k datovým strukturám (1/2)

- Vezměme si třeba dvojsměrně vázaný lineární seznam a souběžné provedení operací DLL_INSERTAFTER a DLL_DELETEAFTER různými vlákny.

- Bez vzájemné synchronizace vláken může seznam skončit v nekonzistentním stavu nebo může dojít k odkazu přes NULL ukazatel.

Souběžný přístup k datovým strukturám (2/2)

- Můžeme před zahájením operací získat zámek pro celý seznam.
- Jenže kde pak máme výhodu paralelního přístupu?!
- Tak co zamknout jenom prvky, se kterými se pracuje?

Souběžný přístup k datovým strukturám (2/2)

- Můžeme před zahájením operací získat zámek pro celý seznam.
- Jenže kde pak máme výhodu paralelního přístupu?!
- Tak co zamknout jenom prvky, se kterými se pracuje?

- Jistě, ale pozor na pořadí zamykání (jinak hrozí **deadlock**).
- A také na efektivitu (paměť pro zámky a operace zamykání navíc).
- Např. v Javě jsou datové struktury, se kterými lze bezpečně pracovat paralelně (thread-safe).
- Také se uvažuje o transakčních pamětech (zkusíme a když se operace nepovede, odvoláme ji a zkusíme později znovu).

Použité zdroje

- Z. Křivka, T. Masopust: Grafové algoritmy. FIT VUT, 2017.
- **http://examples.gurobi.com/traveling-salesman-problem/**
- **http://www.mathematics.pitt.edu/sites/default/files/TSP.pdf**