# Nedeterminismus

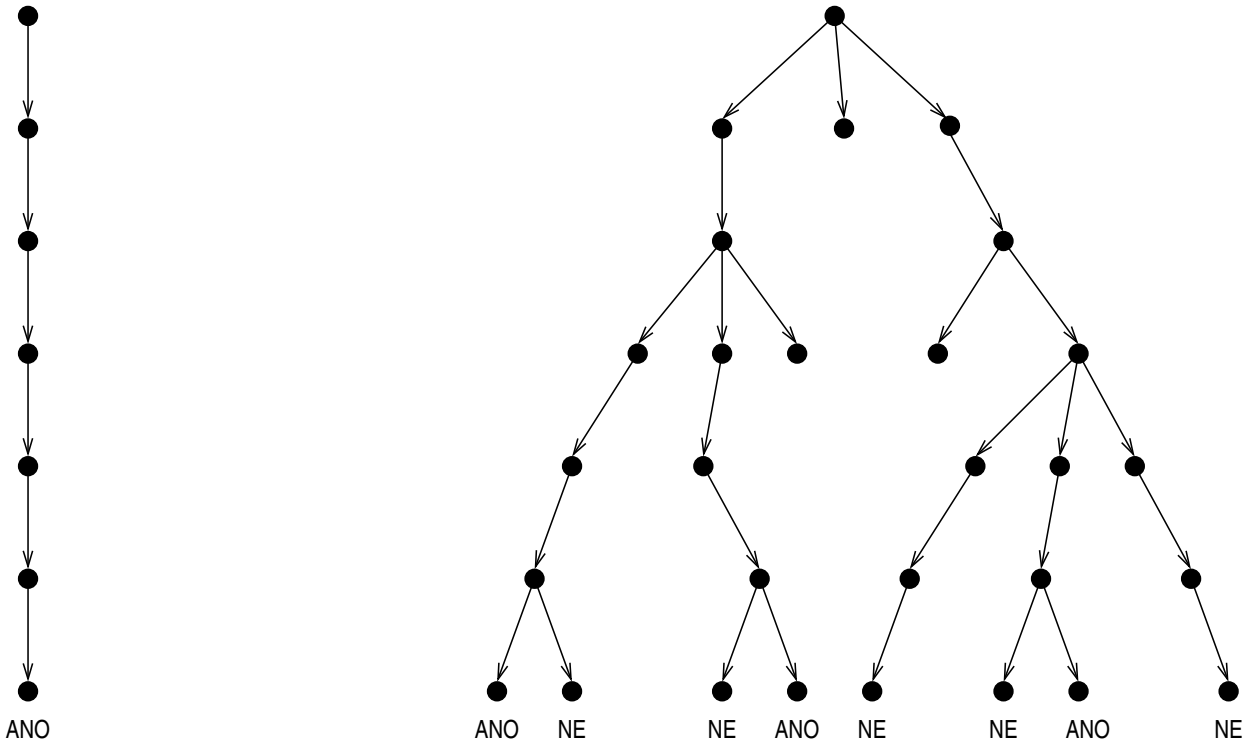
# Nedeterminismus

Nedeterministický stroj RAM:

- Je definován velice podobně jako deterministický RAM.
- Navíc má instrukci NDJUMP x OR y, která umožňuje stroji vybrat si jedno z možných pokračování.
- Pokud ze všech možných výpočtů takového stroje nad zadaným vstupem alespoň jeden skončí s odpovědí ANO, je odpověď ANO.
- Pokud všechny výpočty skončí s odpovědí NE, je odpověď NE.

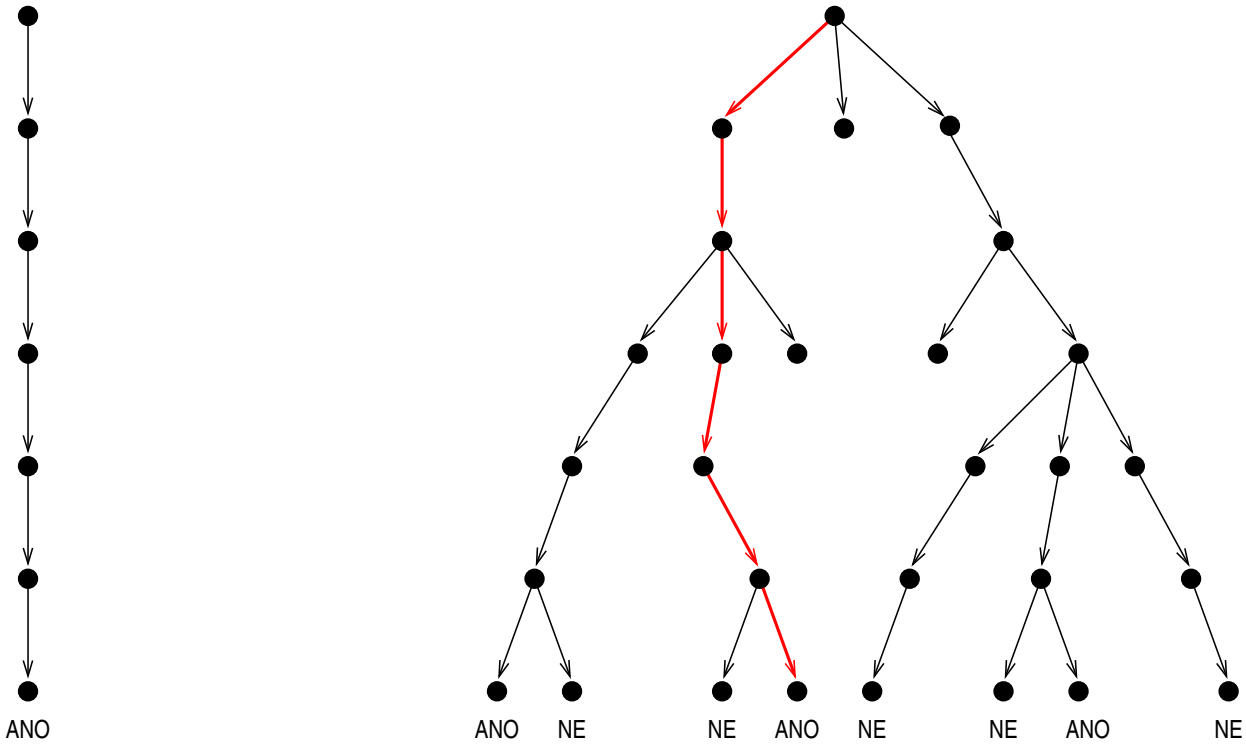
Podobně můžeme definovat nedeterministické verze jiných výpočetních modelů, např. nedeterministické Turingovy stroje.

# Nedeterminismus




- Dobu výpočtu nedeterministického stroje RAM (nebo jiného nedeterministického stroje) nad zadaným vstupem definujeme jako délku nejdelšího možného výpočtu nad tímto vstupem.

# Nedeterminismus



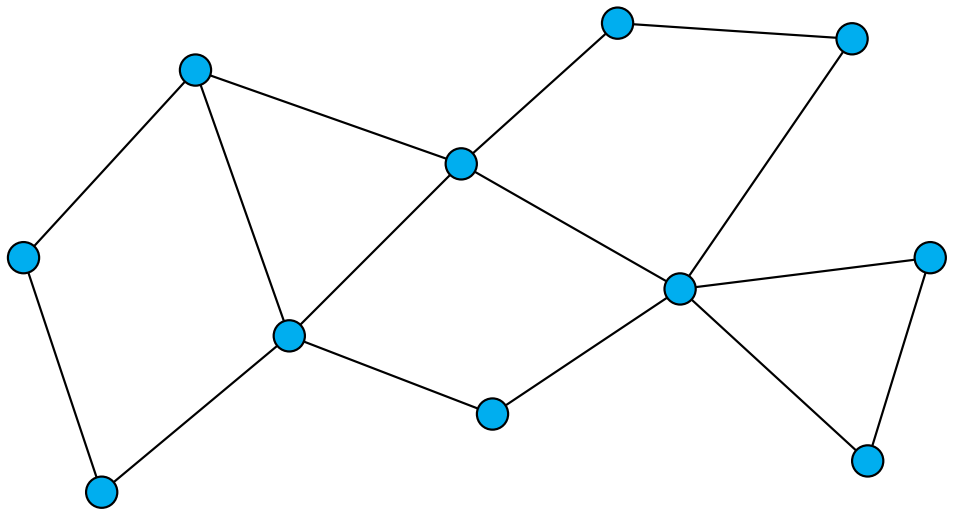

- Dobu výpočtu nedeterministického stroje RAM (nebo jiného nedeterministického stroje) nad zadaným vstupem definujeme jako délku nejdelšího možného výpočtu nad tímto vstupem.

# Nedeterminismus

## Problém „Barvení grafu $k$ barvami“

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Je možné obarvit vrcholy grafu $G$ $k$ barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?



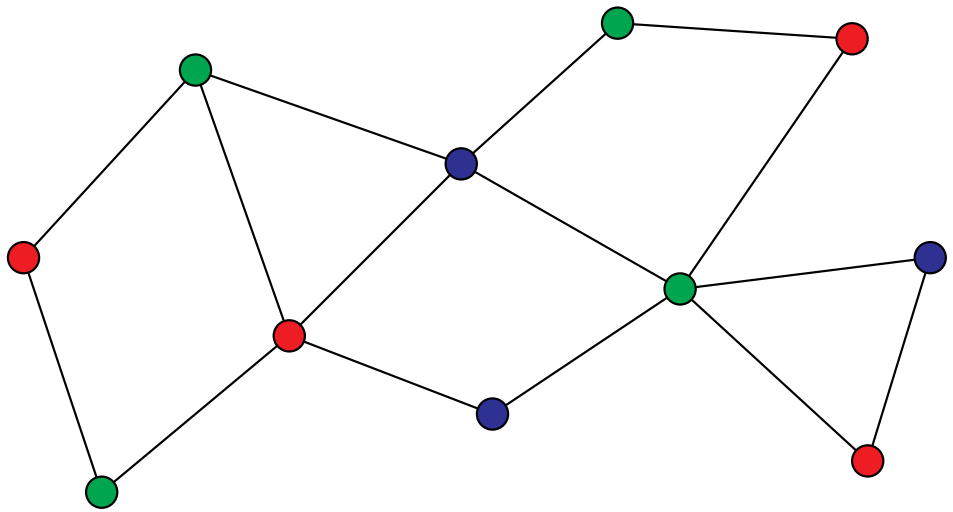
$k = 3$

# Nedeterminismus

## Problém „Barvení grafu $k$ barvami“

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Je možné obarvit vrcholy grafu $G$ $k$ barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?



$k = 3$

# Nedeterminismus

## Problém „Barvení grafu $k$ barvami“

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Je možné obarvit vrcholy grafu $G$ $k$ barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

Nedeterministický algoritmus pracuje následovně:

1. Každému vrcholu grafu $G$ nedeterministicky přiřadí jednu z $k$ barev.
2. Projde všechny hrany grafu $G$ a u každé z nich zkotroluje, že oba její koncové vrcholy jsou obarveny různými barvami. Pokud ne, skončí s odpovědí NE.
3. Pokud prošel všechny hrany a u všech byly koncové vrcholy obarveny různými barvami, skončí s odpovědí ANO.

# Nedeterminismus

## Problém „Isomorfismus grafů“

Vstup: Neorientované grafy $G_1 = (V_1, E_1)$ a $G_2 = (V_2, E_2)$.

Otázka: Jsou grafy $G_1$ a $G_2$ isomorfní?

**Poznámka:** Grafy $G_1$ a $G_2$ jsou isomorfní, jestliže existuje nějaká bijekce $f : V_1 \to V_2$ taková, že pro libovolné dva vrcholy $u, v \in V_1$ platí $(u, v) \in E_1$ právě když $(f(u), f(v)) \in E_2$.

Nedeterministický algoritmus pracuje následovně:

1. Nedeterministicky zvolí hodnoty funkce $f$ pro všechny $v \in V_1$.
2. Deterministicky ověří, že $f$ je bijekce a že pro všechny dvojice vrcholů je splněna výše uvedená podmínka.
3. Pokud je některá z podmínek porušena, skončí s odpovědí NE, v opačném případě s odpovědí ANO.

# Nedeterminismus

Na nedeterminismus můžeme nahlížet dvěma způsoby:

1. Ve chvíli, kdy má stroj nedeterministicky zvolit mezi několika možnostmi, tak „uhodne“, která z těchto možností povede k odpovědi ANO (pokud taková možnost existuje).

2. Ve chvíli, kdy má stroj nedeterministicky zvolit mezi několika možnostmi, rozdělí se do tolika kopií, kolik je těchto možností, a každá z těchto kopií pokračuje ve výpočtu odpovídající jedné z možností, přičemž pracují všechny paralelně.

   Odpověď je ANO právě tehdy, když alespoň jedna z kopií stroje odpoví ANO.

# Nedeterminismus

- Z hlediska rozhodnutelnosti nepřináší nedeterministické algoritmy oproti deterministickým nic dalšího navíc:
  Pokud je nějaký problém možné řešit nedeterministickým strojem RAM nebo TS, tak je ho možné řešit i deterministickým, který postupně vyzkouší všechny možné výpočty nedeterministického stroje nad daným vstupem.
- Nedeterminismus má význam především při zkoumání složitosti problémů.

# Nedeterministické třídy složitosti

## Definice

Pro funkci $f : \mathbb{N} \to \mathbb{N}$ rozumíme **třídou časové složitosti** $\mathcal{NT}(f)$ množinu těch problémů, které jsou řešeny nedeterministickými RAMy s časovou složitostí v $O(f(n))$.

## Definice

Pro funkci $f : \mathbb{N} \to \mathbb{N}$ rozumíme **třídou prostorové složitosti** $\mathcal{NS}(f)$ množinu těch problémů, které jsou řešeny nedeterminictickými RAMy s prostorovou složitostí v $O(f(n))$.

# Třída NPTIME

## Definice

$$\text{NPTIME} = \bigcup_{k=0}^{\infty} \mathcal{NT}(n^k)$$

- NPTIME (někdy se píše jen NP) je třída všech problémů, pro které existuje nedeterministický algoritmus s polynomiání časovou složitostí.
- Do NPTIME tedy patří problémy, u kterých je možné pro daný vstup rychle ověřit, že odpověď je ANO, pokud nám ten, kdo nás o tom chce přesvědčit, dodá nějakou dodatečnou informaci.
- Je zřejmé, že PTIME $\subseteq$ NPTIME, neboť na deterministické algoritmy se můžeme dívat jako na speciální případ nedeterministických.

# Třídy složitosti

- Podobně můžeme definovat třídu NPSPACE.
- Obdobně jako v deterministickém případě zřejmě platí NPTIME $\subseteq$ NPSPACE.
- Nedeterministický RAM můžeme simulovat deterministickým, který zkusí všechny možné výpočty. Každý z možných výpočtů použije maximálně polynomiálně mnoho paměti a jednotlivé výpočty si nic nepředávájí, takže můžeme používat pro všechny stejnou oblast paměti. Platí tedy NPTIME $\subseteq$ PSPACE.
- Je rovněž známo, že pokud je problém rozhodován nedeterministickým Turingovým strojem s prostorovou složitostí $f(n)$, tak je rozhodován i nějakým deterministickým Turingovým strojem s prostorovou složitostí $O((f(n))^2)$. Z toho plyne, že NPSPACE $\subseteq$ PSPACE.

$$\text{PTIME} \subseteq \text{NPTIME} \subseteq \text{PSPACE} = \text{NPSPACE}$$

# NP-úplné problémy

# Polynomiální převod mezi problémy

Mějme nějaké dva rozhodovací problémy $A$ a $B$.

Problém $A$ je **převeditelný** na problém $B$, jestliže existuje algoritmus $P$ takový, že:

- Jako vstup může dostat libovolnou instanci problému $A$.
- K instanci problému $A$, kterou dostane jako vstup (označme ji $x$), vyprodukuje jako svůj výstup instanci problému $B$ (označme ji $P(x)$).
- Platí

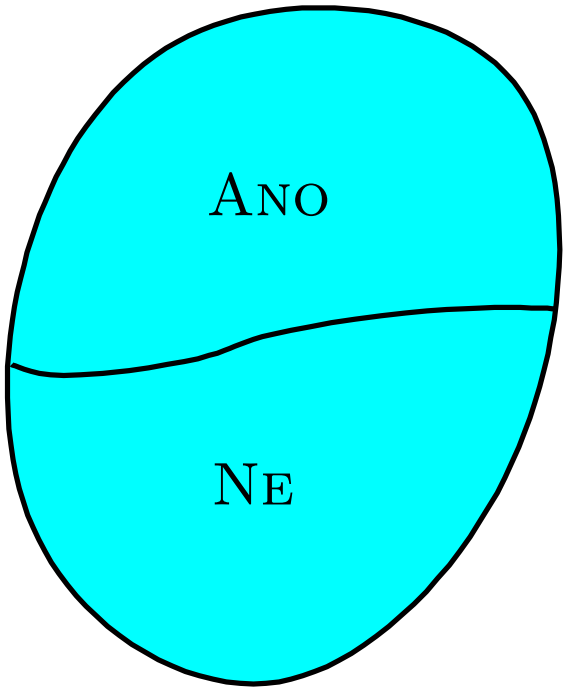
$$x \in A \quad \Leftrightarrow \quad P(x) \in B$$

tj. pro vstup $x$ je v problému $A$ odpověď ANO právě tehdy, když pro vstup $P(x)$ je v problému $B$ odpověď ANO.

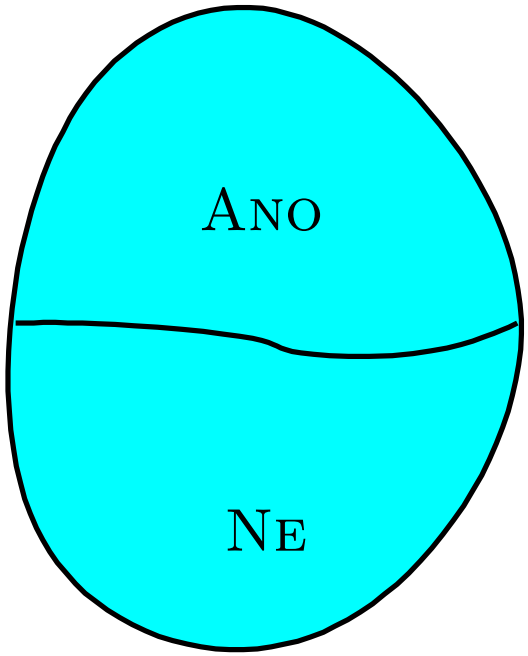
Pokud je navíc algoritmus $P$ polynomiální, pak říkáme, že problém $A$ je **polynomiálně převeditelný** na problém $B$.

# Polynomiální převod mezi problémy

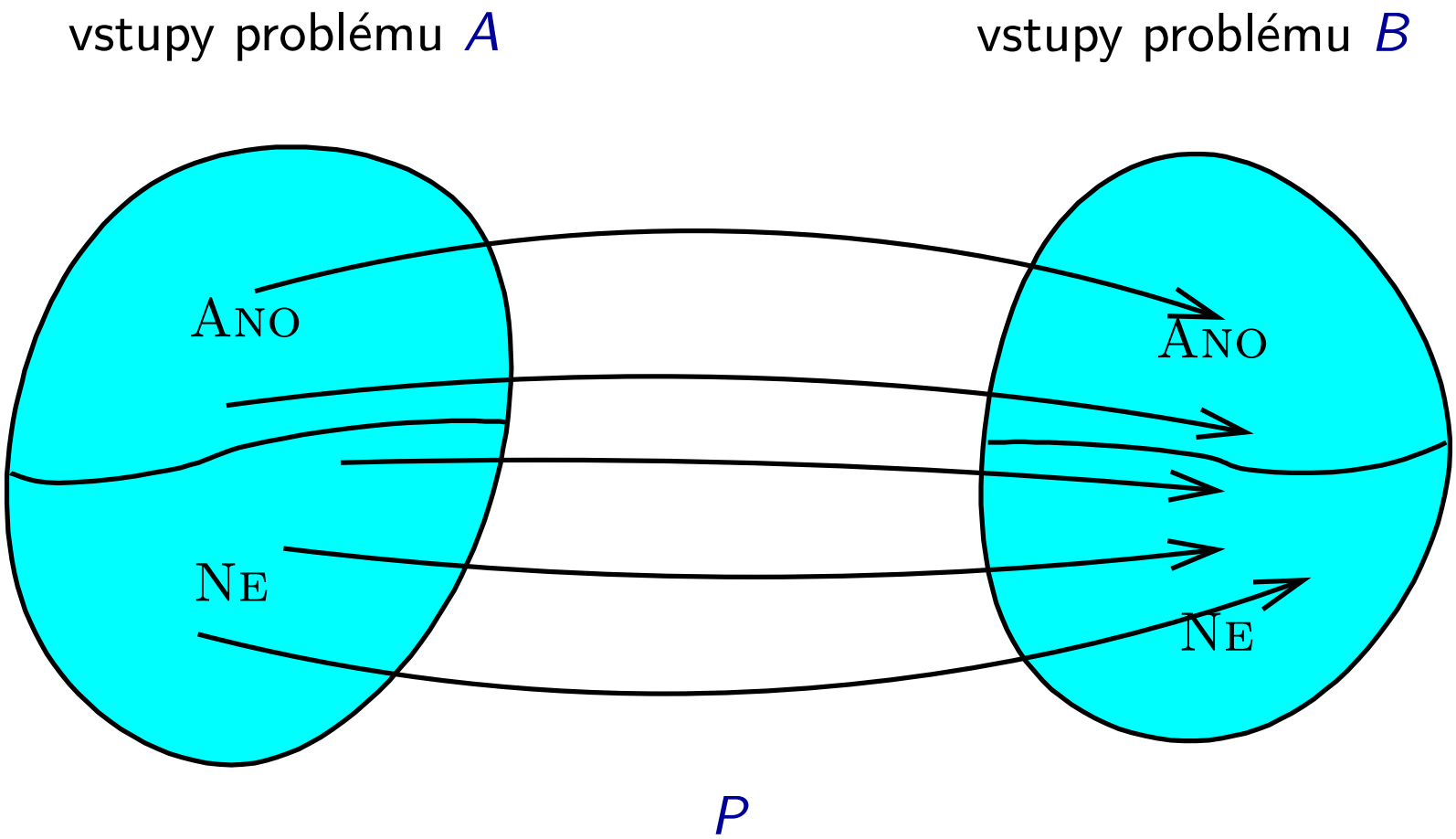
vstupy problému $A$




vstupy problému $B$



# Polynomiální převod mezi problémy



# Polynomiální převod mezi problémy

Řekněme, že problém $A$ je polynomiálně převeditelný na problém $B$, tj. existuje (polynomiální) algoritmus $P$ realizující tento převod.

Pokud je problém $B$ ve třídě PTIME, pak i problém $A$ je ve třídě PTIME.

Řešení problému $A$ pro vstup $x$:

- Zavoláme $P$ se vstupem $x$, vrátí nám hodnotu $P(x)$.
- Zavoláme algoritmus řešící problém $B$ se vstupem $P(x)$. Hodnotu, kterou nám vrátí, vypíšeme jako výsledek.

Z toho plyne:

Pokud $A$ není v PTIME, tak ani $B$ nemůže být v PTIME.

# Problém SAT

Definice problému SAT:

**SAT (splnitelnost booleovských formulí)**

Vstup: Booleovská formule $\varphi$.

Otázka: Je $\varphi$ splnitelná?

**Příklad:**

Formule $\varphi_1 = x_1 \wedge (\neg x_2 \vee x_3)$ je splnitelná:
např. při ohodnocení $\nu$, kde $[x_1]_\nu = 1$, $[x_2]_\nu = 0$, $[x_3]_\nu = 1$, platí $[\varphi_1]_\nu = 1$.

Formule $\varphi_2 = (x_1 \wedge \neg x_1) \vee (\neg x_2 \wedge x_3 \wedge x_2)$ není splnitelná:
pro libovolné ohodnocení $\nu$ platí $[\varphi_2]_\nu = 0$.

# Problém 3-SAT

3-SAT je varianta problému SAT, ve které se omezujeme na formule určitého speciálního typu:

## 3-SAT

Vstup: Formule $\varphi$ v konjunktivní normální formě, kde každá klauzule obsahuje právě 3 literály.

Otázka: Je $\varphi$ splnitelná?

# Problém 3-SAT

Některé pojmy:

- **Literál** je formule tvaru $x$ nebo $\neg x$, kde $x$ je booleovská proměnná.
- **Klauzule** je disjunkce literálů.

  Příklady: $x_1 \vee \neg x_2$ $\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}$ $x_6$
- Formule je v **konjunktivní normální formě (KNF)**, jestliže je konjunkcí klauzulí.

  Příklad: $(x_1 \vee \neg x_2) \wedge (\neg x_5 \vee x_8 \vee \neg x_{15} \vee \neg x_{23}) \wedge x_6$

V případě problému 3-SAT tedy vyžadujeme, aby formule $\varphi$ byla v KNF a navíc, aby každá klauzule obsahovala právě tři literály.

**Příklad:**

$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$

# Problém 3-SAT

Následující formule je splnitelná:

$$(x_1 \vee \neg x_2 \vee x_4) \wedge (\neg x_1 \vee x_3 \vee x_3) \wedge (\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee \neg x_3 \vee x_4)$$

Např. při ohodnocení $\nu$, kde

$$[x_1]_\nu = 0$$
$$[x_2]_\nu = 1$$
$$[x_3]_\nu = 0$$
$$[x_4]_\nu = 1$$

je $[\varphi_1]_\nu = 1$.

Naproti tomu následující formule není splnitelná:

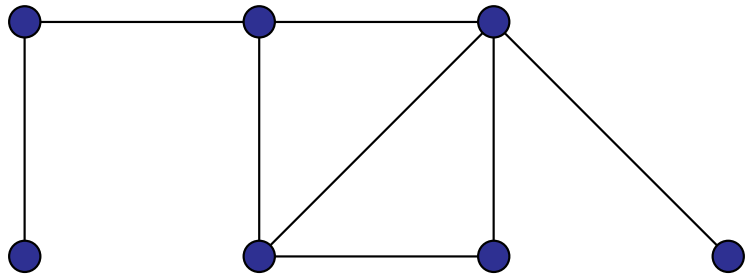
$$(x_1 \vee x_1 \vee x_1) \wedge (\neg x_1 \vee \neg x_1 \vee \neg x_1)$$

# Problém nezávislé množiny (IS)

## Problém nezávislé množiny (IS)

Vstup: Neorientovaný graf $G$, číslo $k$.

Otázka: Existuje v grafu $G$ nezávislá množina velikosti $k$?



$k = 4$

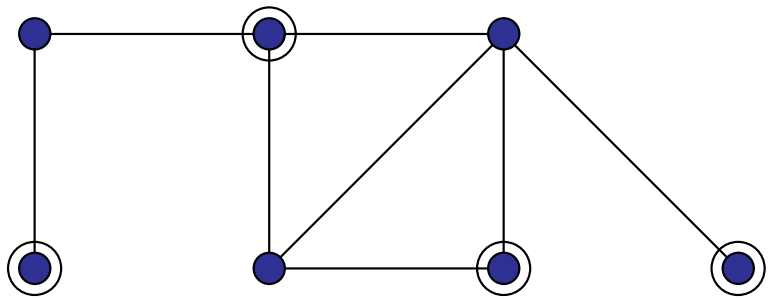
**Poznámka:** **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

# Problém nezávislé množiny (IS)

## Problém nezávislé množiny (IS)

Vstup: Neorientovaný graf $G$, číslo $k$.

Otázka: Existuje v grafu $G$ nezávislá množina velikosti $k$?

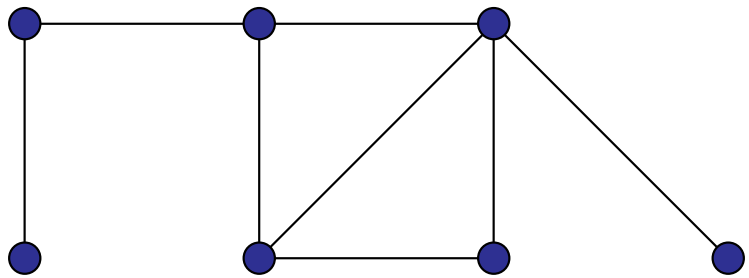


$k = 4$

**Poznámka:** **Nezávislá množina** v grafu je podmnožina vrcholů grafu taková, že žádné dva vrcholy z této podmnožiny nejsou spojeny hranou.

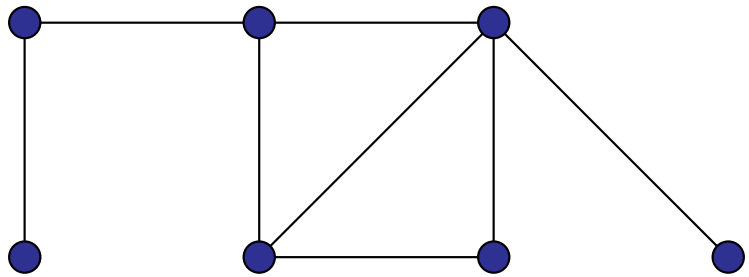
# Problém nezávislé množiny (IS)

Příklad instance, kde je odpověď Ano:



$k = 4$

Příklad instance, kde je odpověď Ne:



$k = 5$

# Převod 3-SAT na IS

Popíšeme (polynomiální) algoritmus, který bude mít následující vlastnosti:

- **Vstup:** Libovolná instance problému 3-SAT, tj. formule $\varphi$ v konjunktivní normální formě, kde každá klauzule obsahuje právě tři literály.
- **Výstup:** Instance problému IS, tj. neorientovaný graf $G$ a číslo $k$.
- Navíc bude pro libovolný vstup (tj. pro libovolnou formuli $\varphi$ ve výše uvedeném tvaru) zaručeno následující:

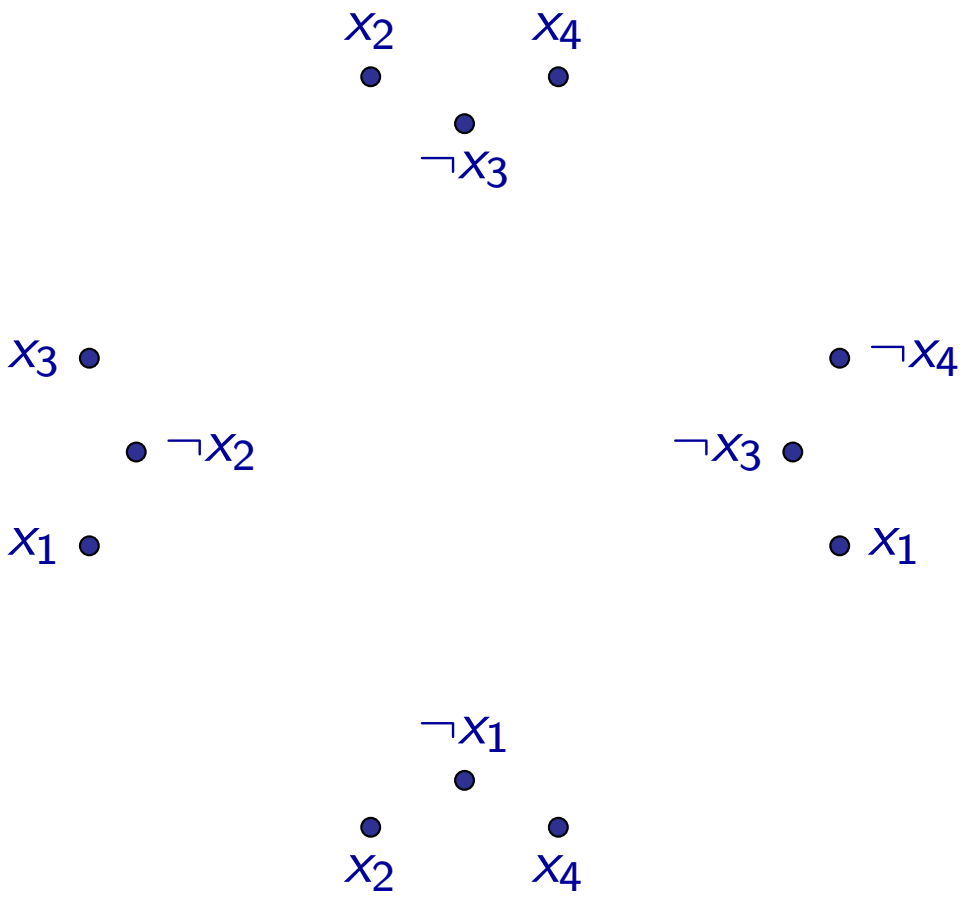
  V grafu $G$ bude existovat nezávislá množina velikosti $k$ právě tehdy, když formule $\varphi$ bude splnitelná.

# Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

# Převod 3-SAT na IS

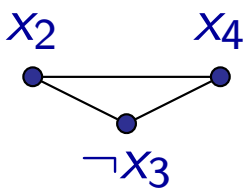
$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

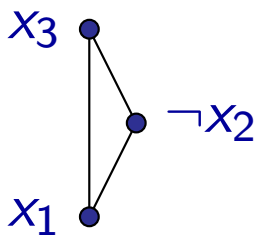


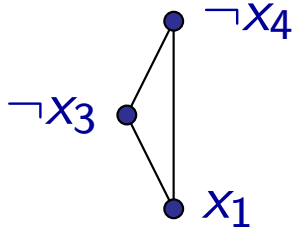

Pro každý literál přidáme do grafu jeden vrchol.

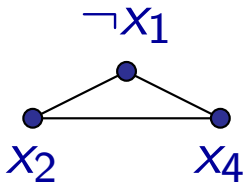
# Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$



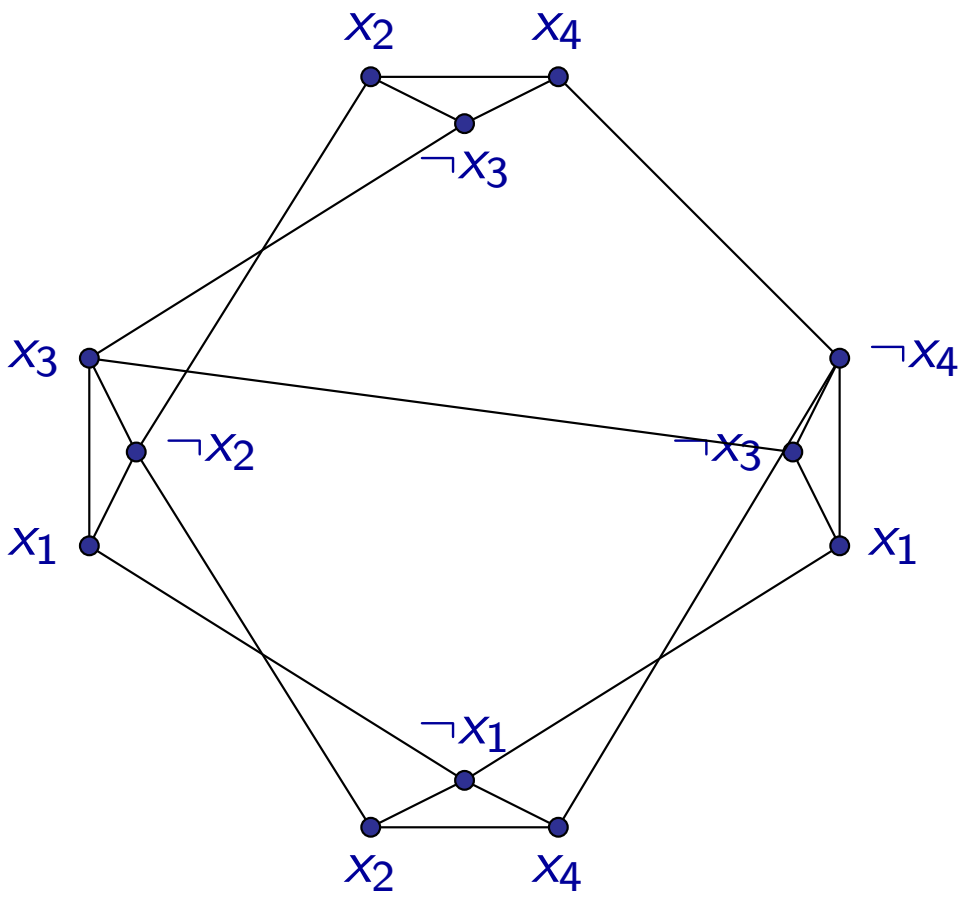




Vrcholy odpovídající literálům patřícím do stejné klauzule spojíme hranami.

# Převod 3-SAT na IS

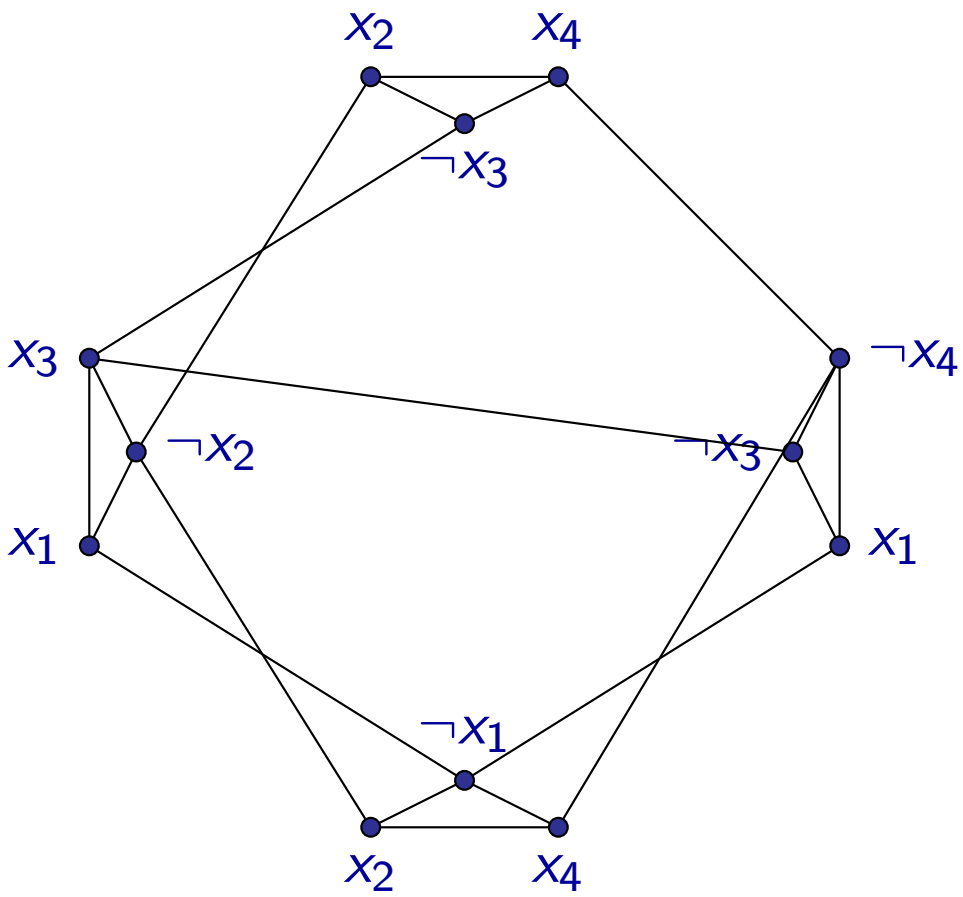
$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$




Dvojice vrcholů odpovídající literálům $x_i$ a $\neg x_i$ spojíme hranami.

# Převod 3-SAT na IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

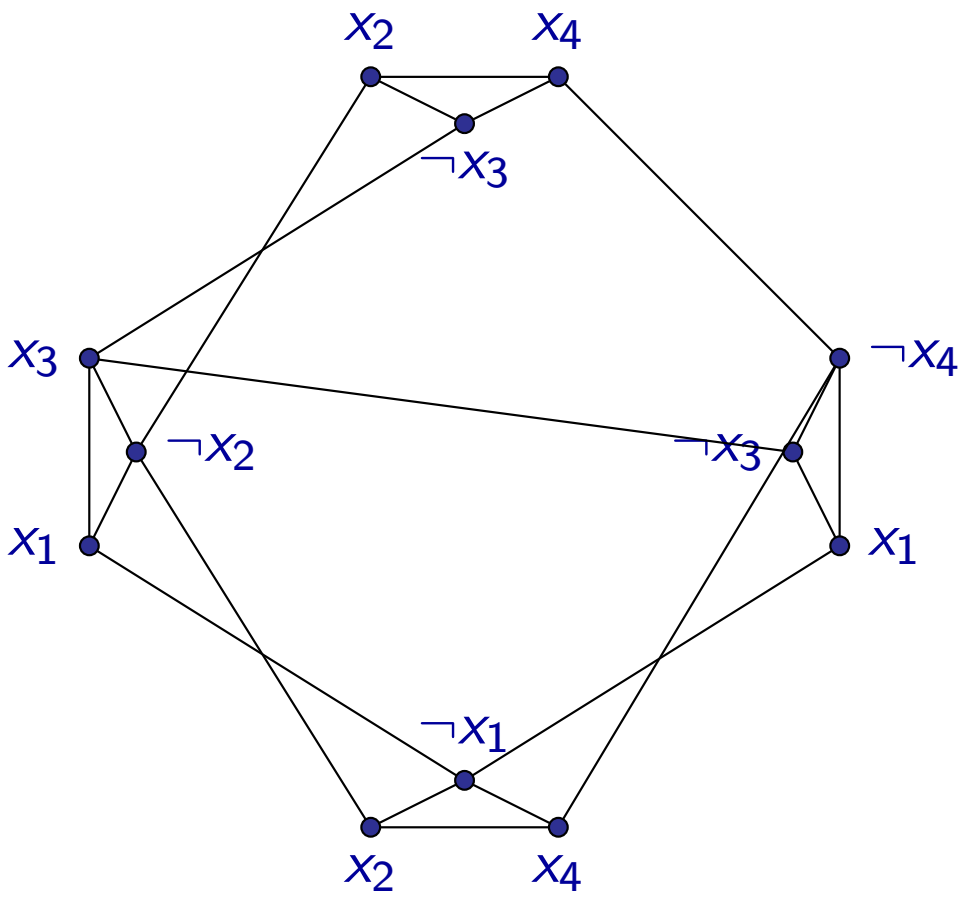



$k = 4$

Číslo $k$ položíme rovno počtu klauzulí.

# Převod 3-SAT na IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$



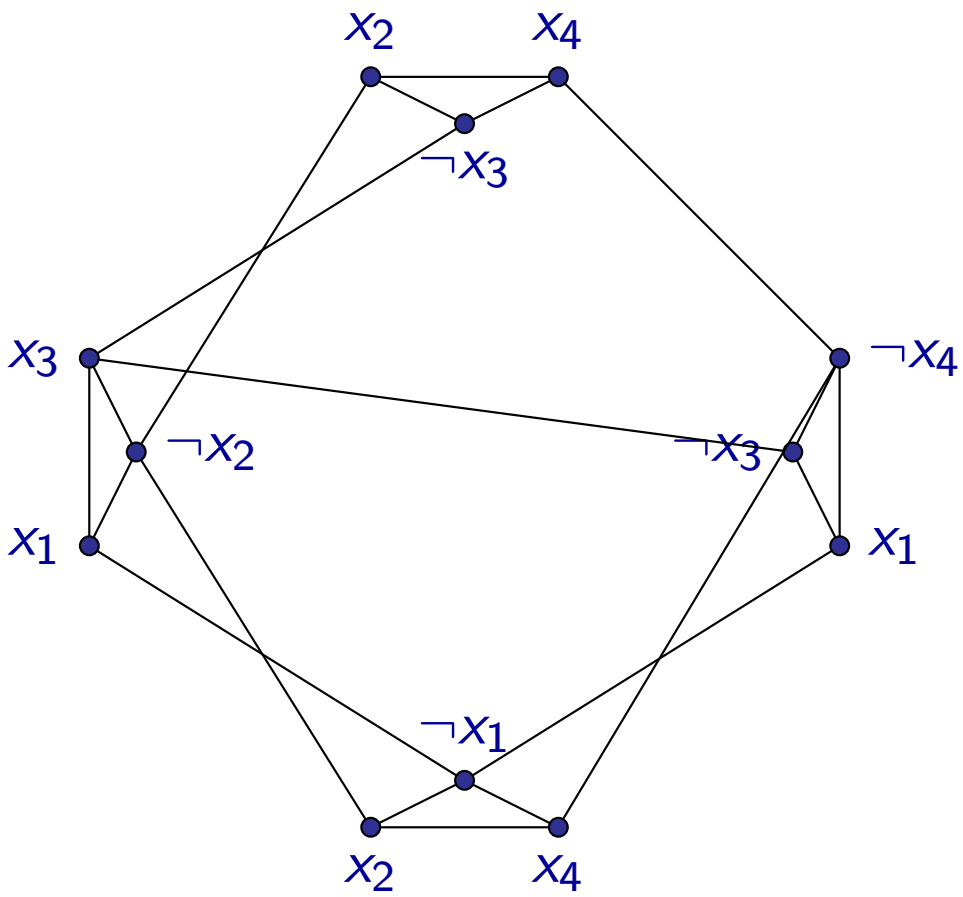

$k = 4$

Vytvořený graf a číslo $k$ vydá algoritmus jako výstup.

# Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

$\nu(x_1) = 1$
$\nu(x_2) = 1$
$\nu(x_3) = 0$
$\nu(x_4) = 1$



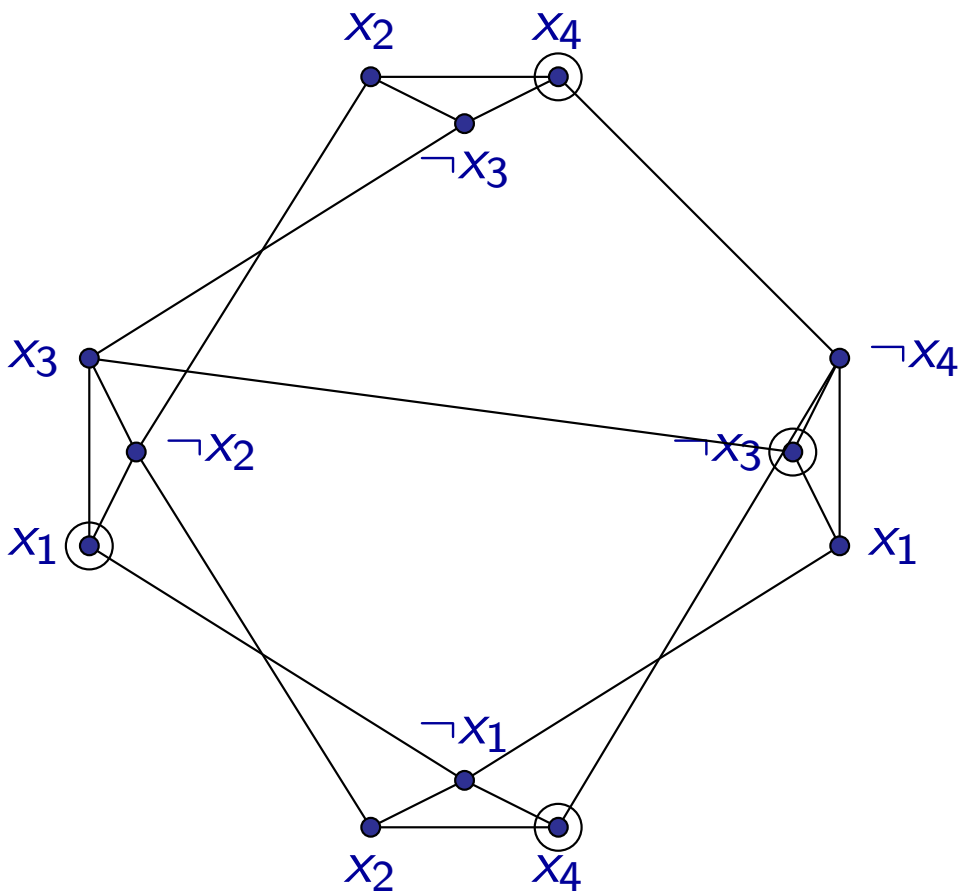

$k = 4$

Jestliže je formule $\varphi$ splnitelná, existuje ohodnocení $\nu$, při kterém má v každé klauzuli alespoň jeden literál hodnotu 1.

# Převod 3-SAT na IS

$$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$$

$\nu(x_1) = 1$
$\nu(x_2) = 1$
$\nu(x_3) = 0$
$\nu(x_4) = 1$



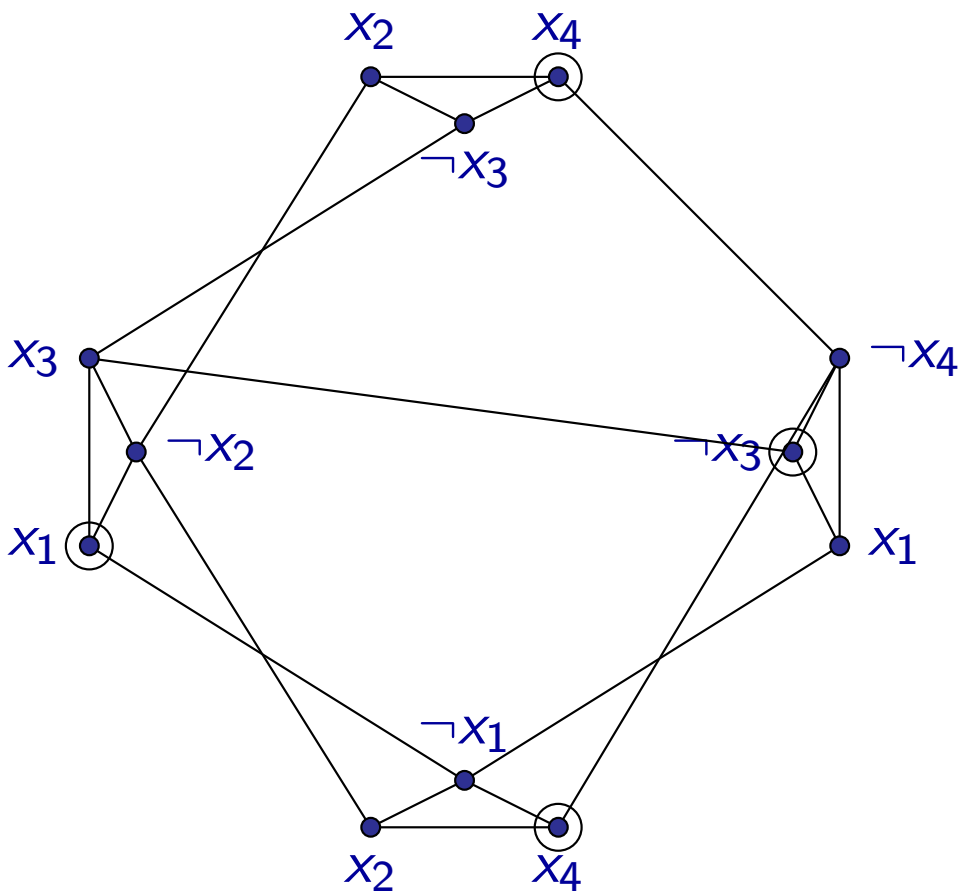

$k = 4$

Z každé klauzule vybereme jeden literál, který má při ohodnocení $\nu$ hodnotu 1, a do nezávislé množiny přidáme odpovídající vrchol.

# Převod 3-SAT na IS

$(x_1 \vee \neg x_2 \vee x_3) \wedge (x_2 \vee \neg x_3 \vee x_4) \wedge (x_1 \vee \neg x_3 \vee \neg x_4) \wedge (\neg x_1 \vee x_2 \vee x_4)$

$\nu(x_1) = 1$
$\nu(x_2) = 1$
$\nu(x_3) = 0$
$\nu(x_4) = 1$




$k = 4$

Lehce ověříme, že vybrané vrcholy tvoří nezávislou množinu.

# Převod 3-SAT na IS

Vybrané vrcholy tvoří nezávislou množinu, protože:

- Z každé trojice vrcholů odpovídající jedné klauzuli byl vybrán jen jeden vrchol.
- Nemohly být současně vybrány vrcholy označené $x_i$ a $\neg x_i$.
  (Při daném ohodnocení $\nu$ má hodnotu $1$ jen jeden z nich.)

# Převod 3-SAT na IS

Na druhou stranu, pokud v grafu $G$ existuje nezávislá množina velikosti $k$, musí určitě splňovat následující vlastnosti:

- Z každé trojice vrcholů odpovídající jedné klauzuli musí být vybrán nejvýše jeden vrchol.

  Protože je ale klauzulí $k$ a je vybráno $k$ vrcholů, musí být z každé takové trojice vybrán právě jeden.

- Nemohly být současně vybrány vrcholy označené $x_i$ a $\neg x_i$.

Ohodnocení tedy zvolíme podle vybraných vrcholů, protože z předchozího vyplývá, že nehrozí, že by neexistovalo.
(Zbylým proměnným přiřadíme libovolné hodnoty.)

Při daném ohodnocení má formule $\varphi$ určitě hodnotu 1, neboť v každé klauzuli má hodnotu 1 alespoň jeden literál.

# Převod 3-SAT na IS

Popsaný algoritmus je určitě polynomiální:

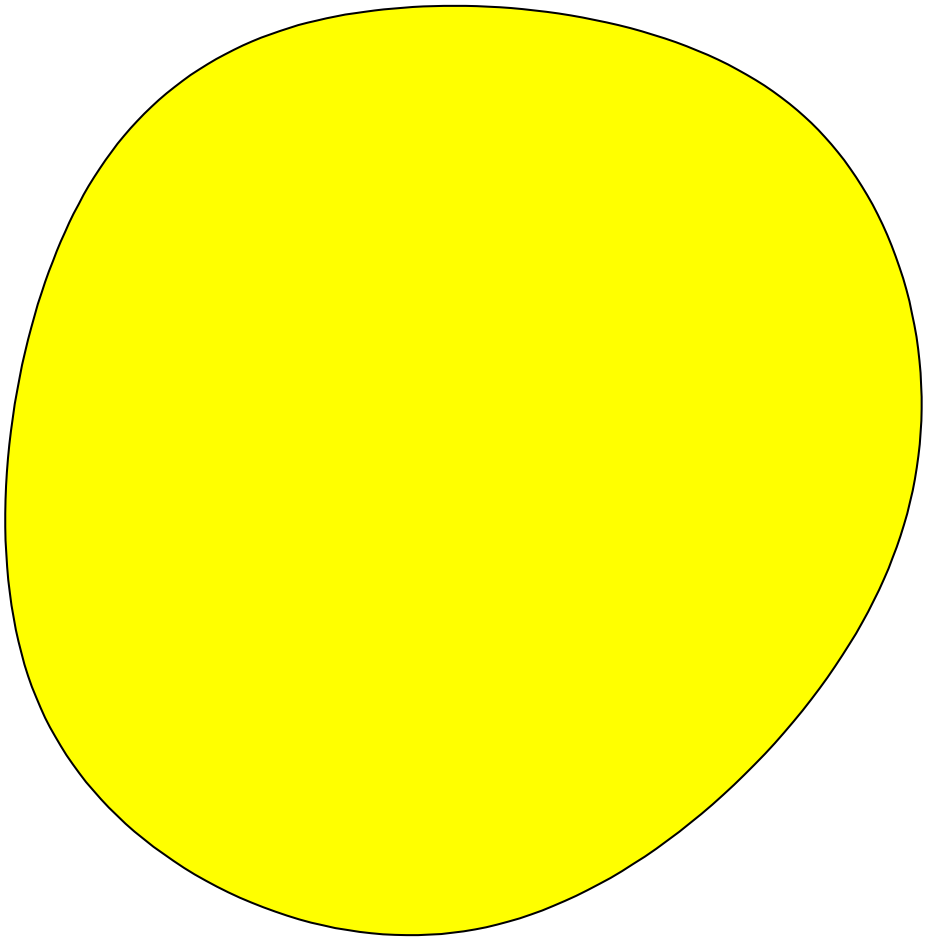
Graf $G$ a číslo $k$ je možné zkonstruovat k formuli $\varphi$ v čase $O(n^2)$, kde $n$ je velikost formule $\varphi$.

Navíc jsme viděli, že ve zkonstruovaném grafu $G$ existuje nezávislá množina velikosti $k$ právě tehdy, když formule $\varphi$ je splnitelná.

Popsaný algoritmus tedy ukazuje, že problém 3-SAT je polynomiálně převeditelný na problém IS.

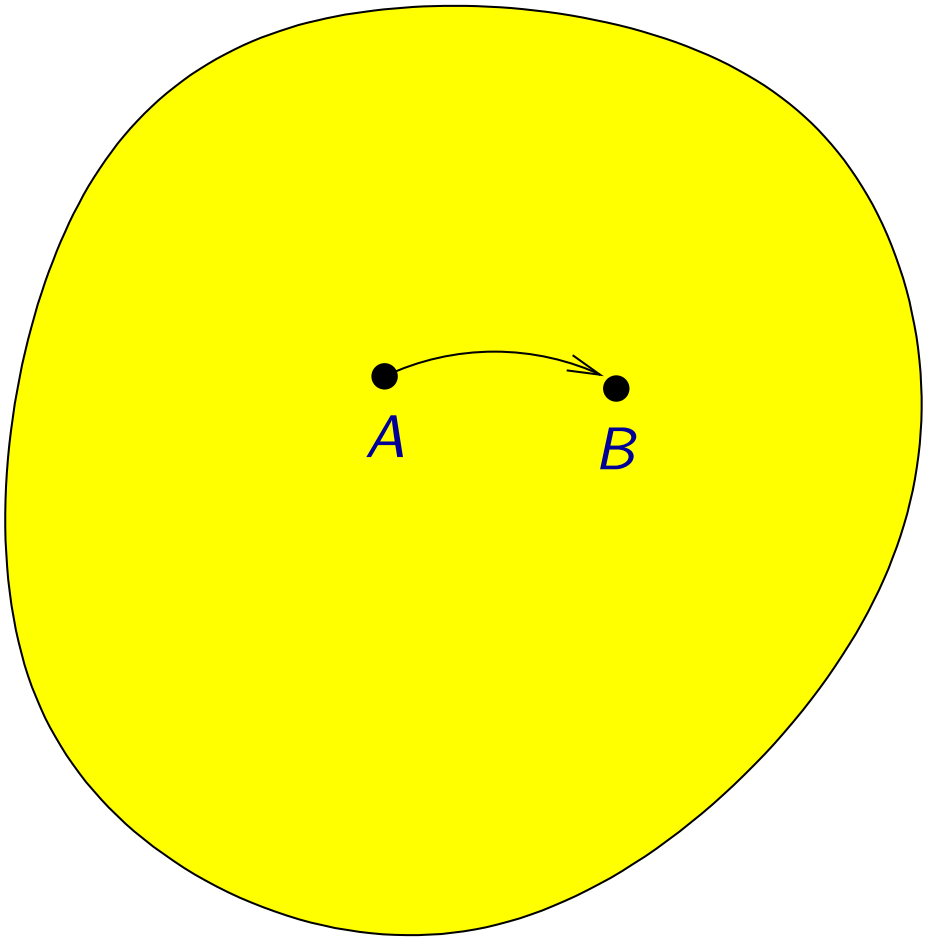
# NP-úplné problémy

Vezměme si množinu všech možných rozhodovacích problémů.



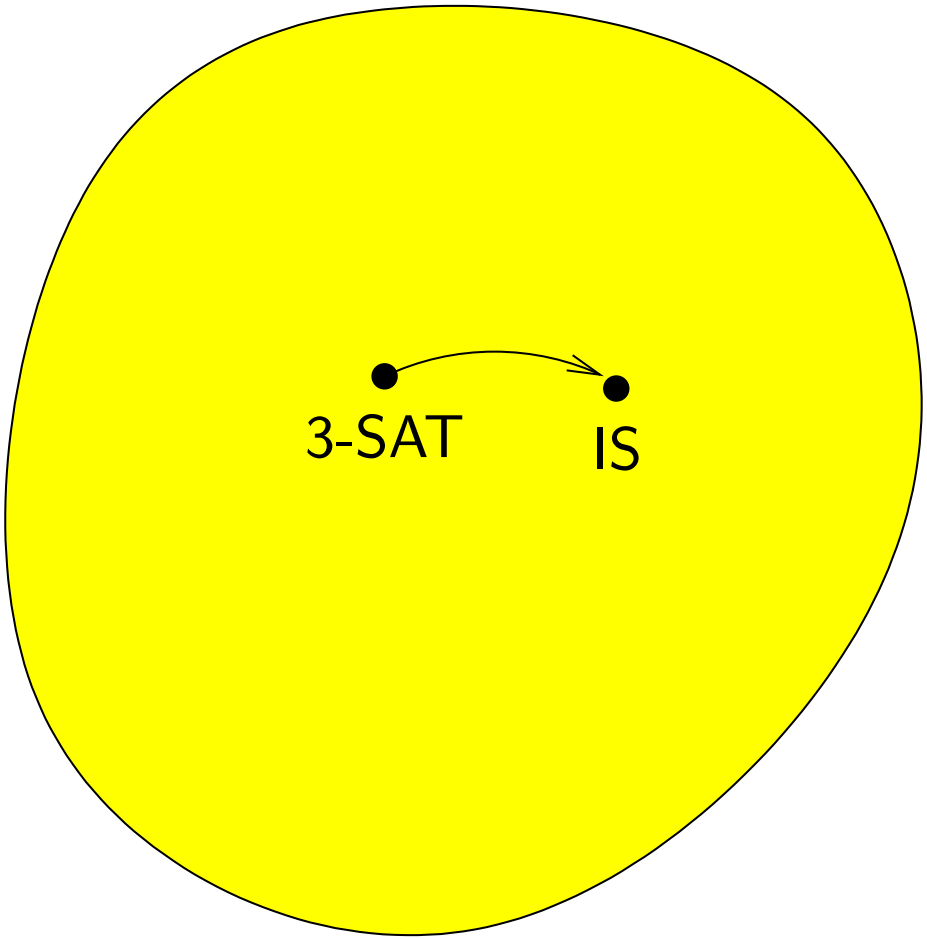
# NP-úplné problémy

Šipkou si znázorníme, že problém $A$ je polynomiálně převeditelný na problém $B$.



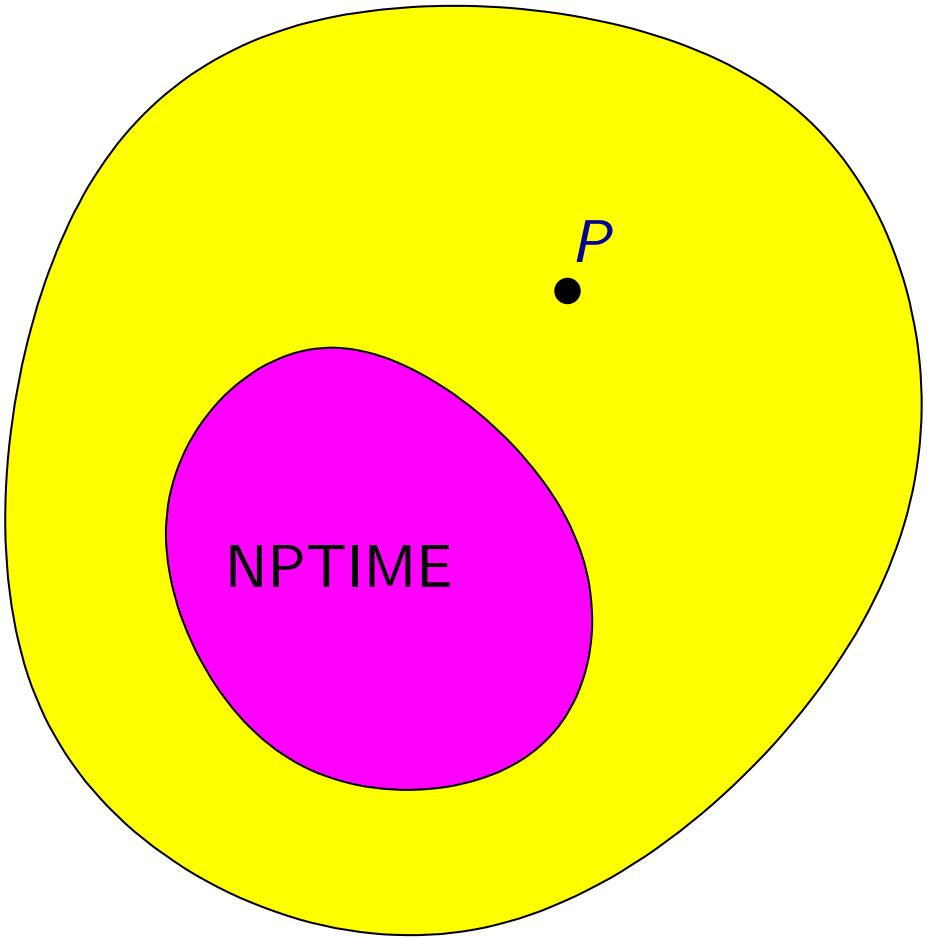
# NP-úplné problémy

Například problém 3-SAT je polynomiálně převeditelný na problém IS.



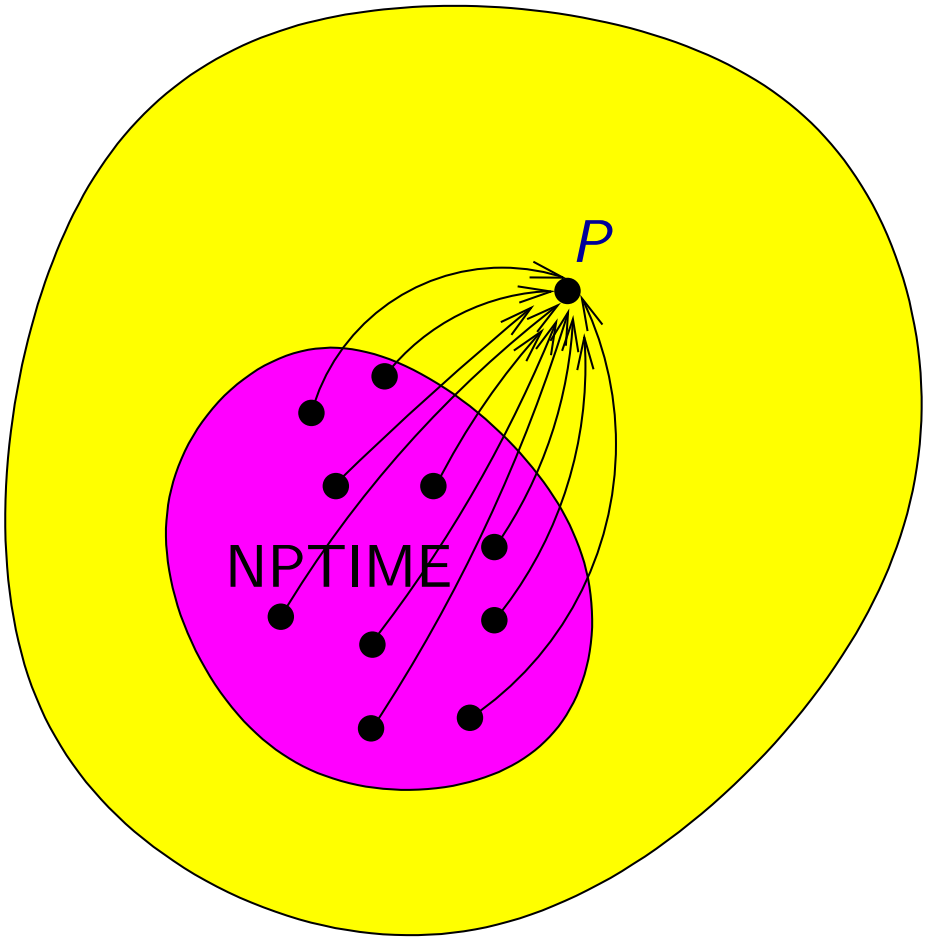
# NP-úplné problémy

Vezměme si nyní třídu NPTIME a nějaký problém $P$.



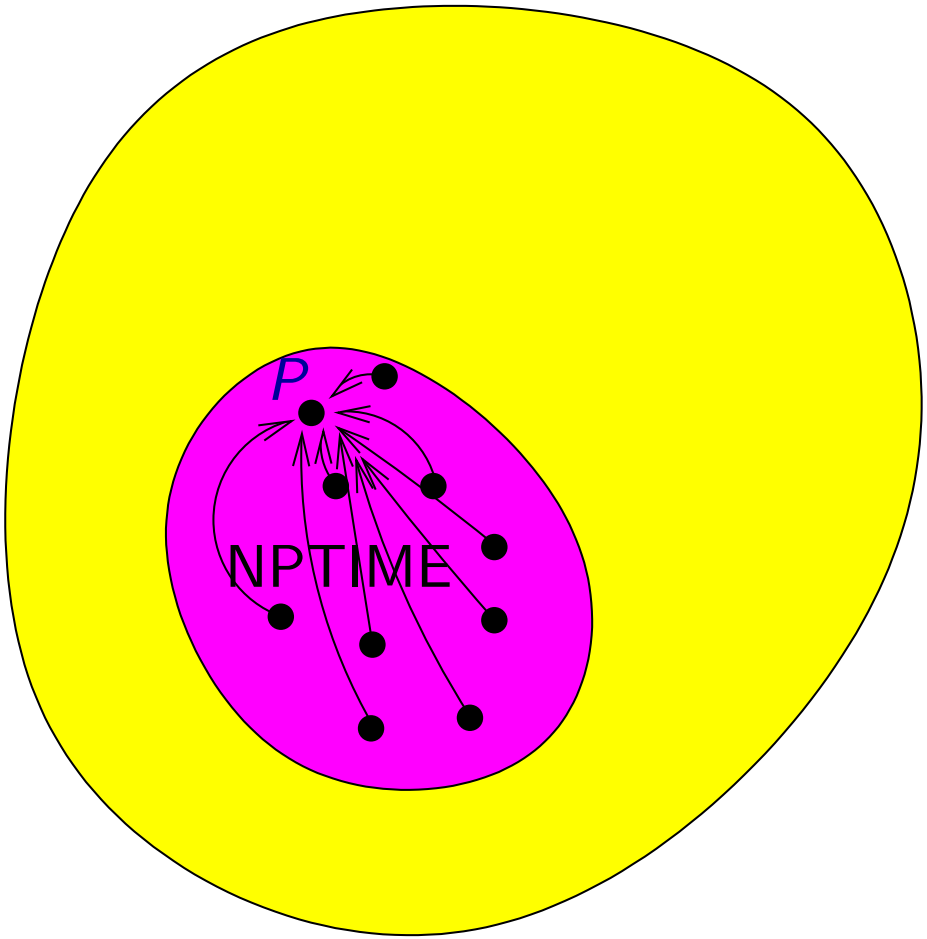
# NP-úplné problémy

Problém $P$ je NP-**těžký**, jestliže každý problém z NPTIME je polynomiálně převeditelný na $P$.



# NP-úplné problémy

Problém $P$ je NP-**úplný**, jestliže je NP-těžký a navíc sám patří do třídy NPTIME.



# NP-úplné problémy

Pokud bychom pro nějaký NP-těžký problém $P$ nalezli polynomiální algoritmus, získali bychom tím polynomiální algoritmus pro každý problém $P'$ z NPTIME:

- Na vstup problému $P'$ bychom nejprve aplikovali algoritmus realizující polynomiální převod z $P'$ na $P$.
- Na vytvořenou instanci problému $P$ bychom aplikovali polynomiální algoritmus řešící problém $P$ a výsledek bychom vrátili jako odpověď pro danou instanci problému $P'$.

V takovém případě by tedy platilo PTIME $=$ NPTIME, neboť pro každý problém z NPTIME by existoval polynomiální (deterministický) algoritmus.

# NP-úplné problémy

Na druhou stranu, pokud existuje alespoň jeden problém z NPTIME, pro který neexistuje polynomiální algoritmus, tak z předchozího plyne, že pro žádný NP-těžký problém nemůže existovat polynomiální algoritmus.

Zda platí první nebo druhá možnost, je otevřený problém.

# NP-úplné problémy

Není těžké si rozmyslet následující:

Pokud je problém $A$ polynomiálně převeditelný na problém $B$ a problém $B$ je polynomiálně převeditelný na problém $C$, pak problém $A$ je polynomiálně převeditelný na problém $C$.

Pokud tedy o nějakém problému $P$ víme, že je NP-těžký a že $P$ je polynomiálně převeditelný na problém $P'$, pak víme, že i problém $P'$ je NP-těžký.

# NP-úplné problémy

**Věta**

Problém SAT je NP-úplný.

Dá se ukázat, že SAT je polynomiálně převeditelný na 3-SAT a viděli jsme, že 3-SAT je polynomiálně převeditelný na IS.

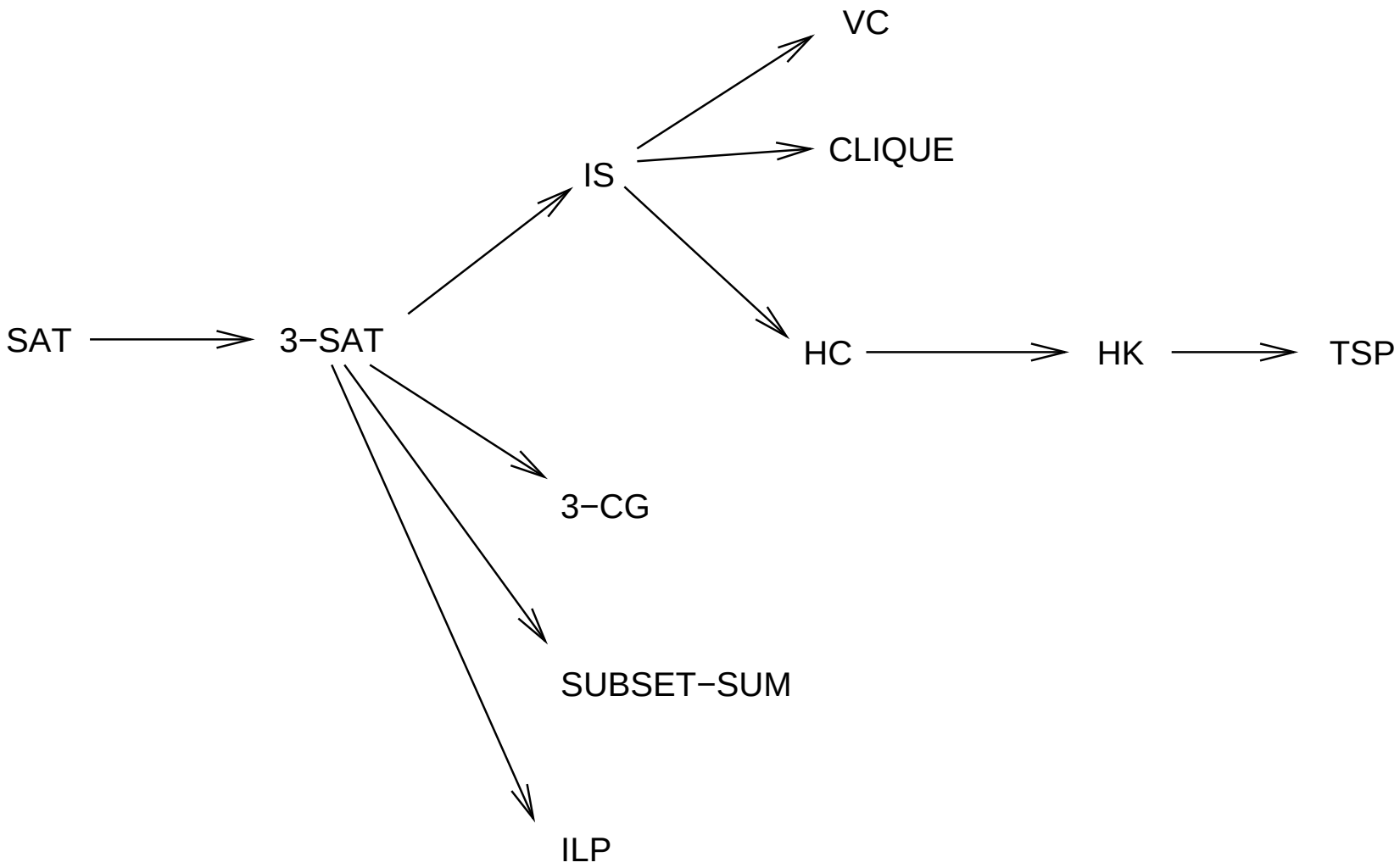
Z toho plyne, že problémy 3-SAT a IS jsou NP-těžké.

To, že 3-SAT i IS jsou v NPTIME je očividné.

Problémy 3-SAT i IS jsou NP-úplné.

# NP-úplné problémy

Polynomiálními převody z již známých NP-úplných problémů se dá ukázat NP-obtížnost celé řady různých dalších problémů:



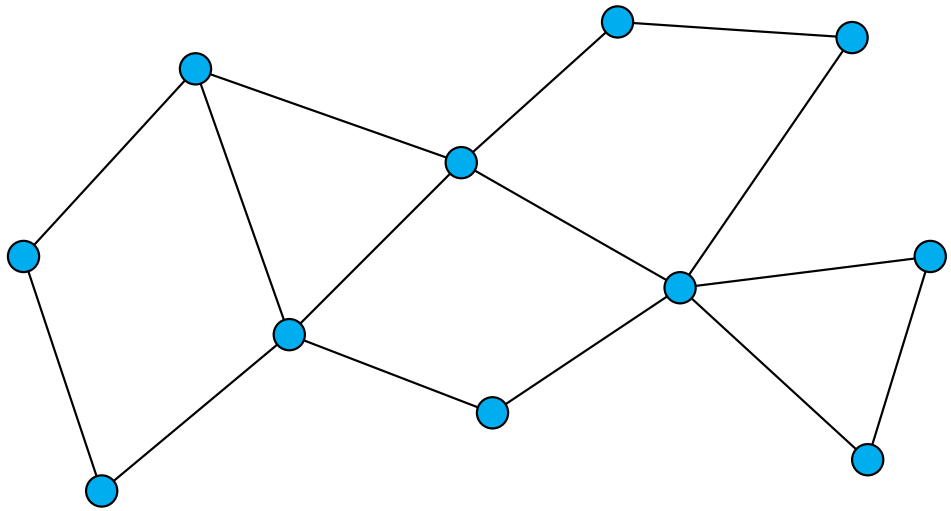
# 3-CG – Barvení grafu 3 barvami

## 3-CG - Problém „Barvení grafu 3 barvami“

Vstup: Neorientovaný graf $G$

Otázka: Lze vrcholy grafu $G$ obarvit 3 barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

**Příklad:**



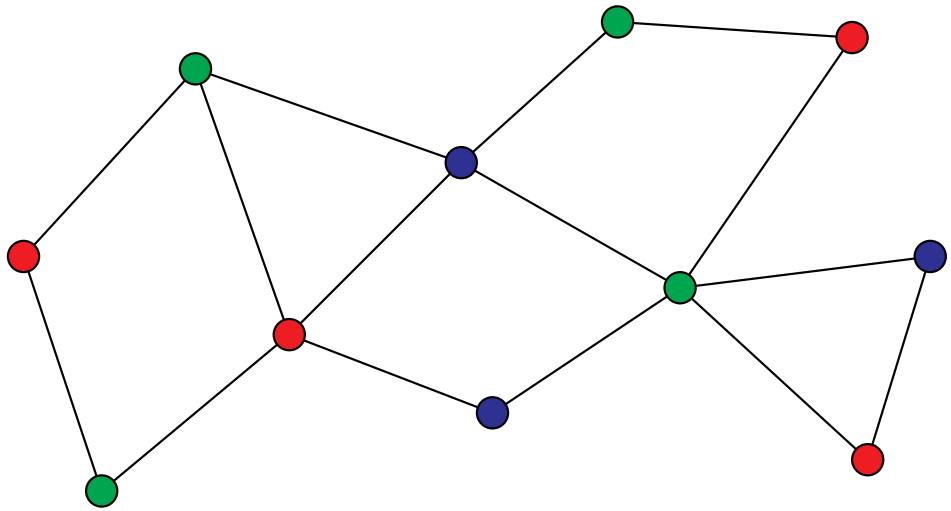
# 3-CG – Barvení grafu 3 barvami

## 3-CG - Problém „Barvení grafu 3 barvami“

Vstup: Neorientovaný graf $G$

Otázka: Lze vrcholy grafu $G$ obarvit 3 barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

**Příklad:**



Odpověď: Ano

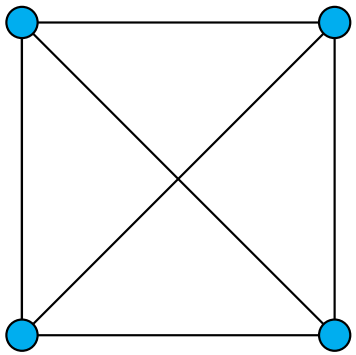
# 3-CG – Barvení grafu 3 barvami

## 3-CG - Problém „Barvení grafu 3 barvami“

Vstup: Neorientovaný graf $G$

Otázka: Lze vrcholy grafu $G$ obarvit 3 barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

**Příklad:**



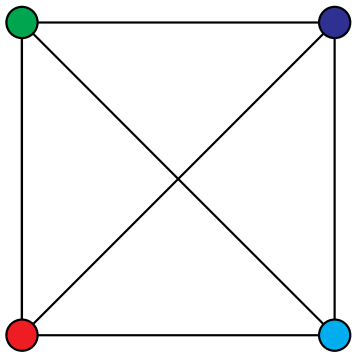
# 3-CG – Barvení grafu 3 barvami

## 3-CG - Problém „Barvení grafu 3 barvami“

Vstup: Neorientovaný graf $G$

Otázka: Lze vrcholy grafu $G$ obarvit 3 barvami tak, aby žádné dva vrcholy spojené hranou neměly stejnou barvu?

**Příklad:**



Odpověď: NE

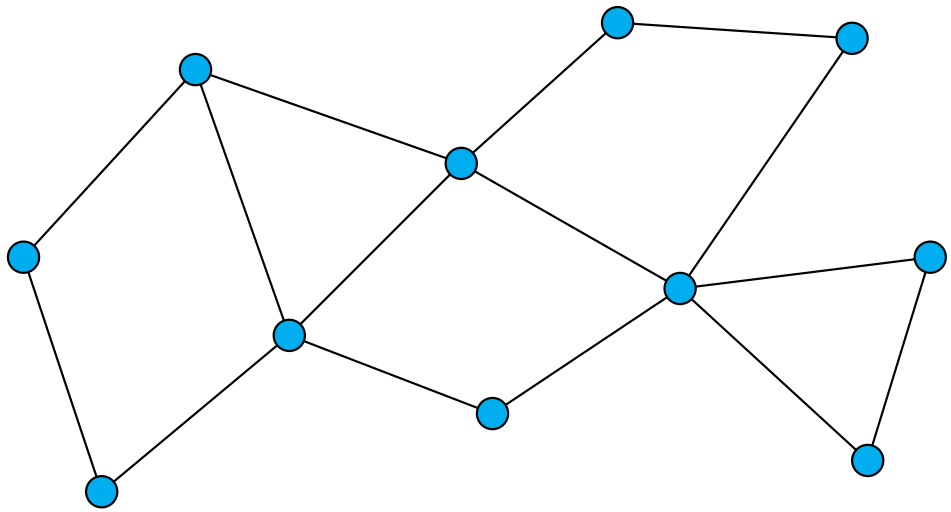
# IS – Nezávislá množina

## IS - Problém „Nezávislá množina“ (independent set)

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Existuje v grafu $G$ nezávislá množina velikosti $k$ (množina $k$ vrcholů, které vzájemně nejsou propojeny hranou)?

**Příklad:** $k = 5$



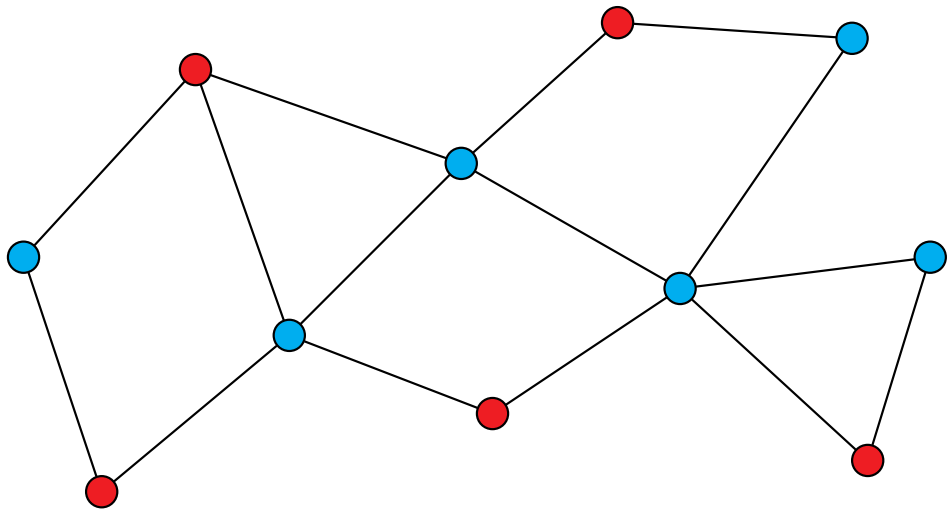
# IS – Nezávislá množina

## IS - Problém „Nezávislá množina“ (independent set)

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Existuje v grafu $G$ nezávislá množina velikosti $k$ (množina $k$ vrcholů, které vzájemně nejsou propojeny hranou)?

**Příklad:** $k = 5$



Odpověď: Ano

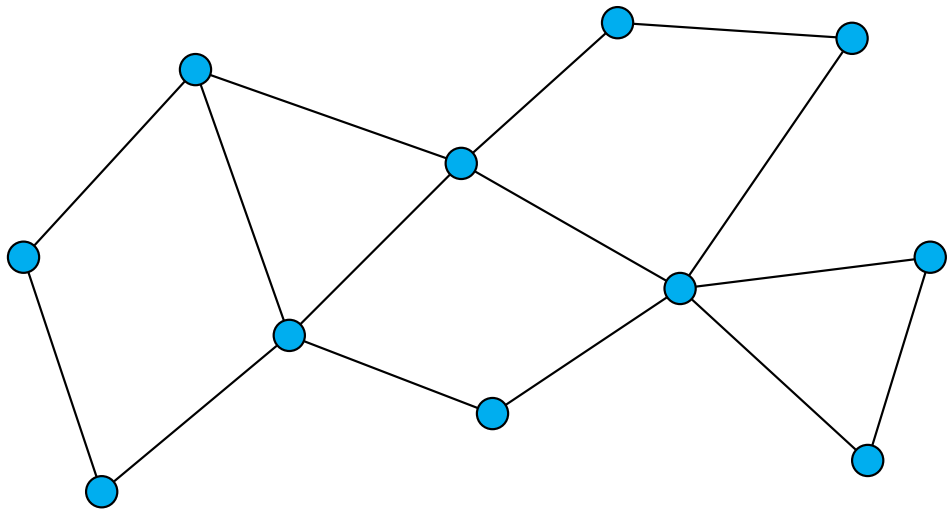
# VC – Vrcholové pokrytí

## VC – vrcholové pokrytí (vertex cover)

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Existuje v grafu $G$ množina vrcholů velikosti $k$ taková, že každá hrana má alespoň jeden svůj vrchol v této množině?

**Příklad:** $k = 6$



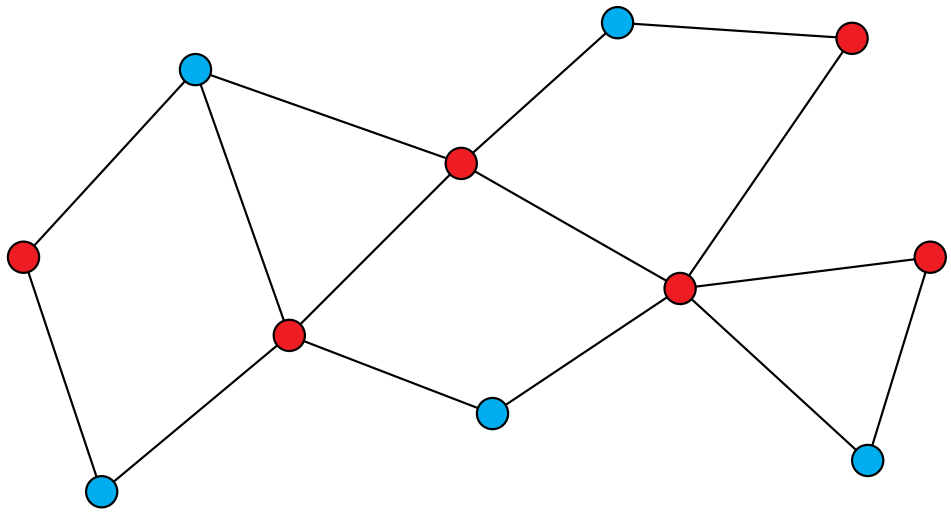
# VC – Vrcholové pokrytí

## VC – vrcholové pokrytí (vertex cover)

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Existuje v grafu $G$ množina vrcholů velikosti $k$ taková, že každá hrana má alespoň jeden svůj vrchol v této množině?

**Příklad:** $k = 6$



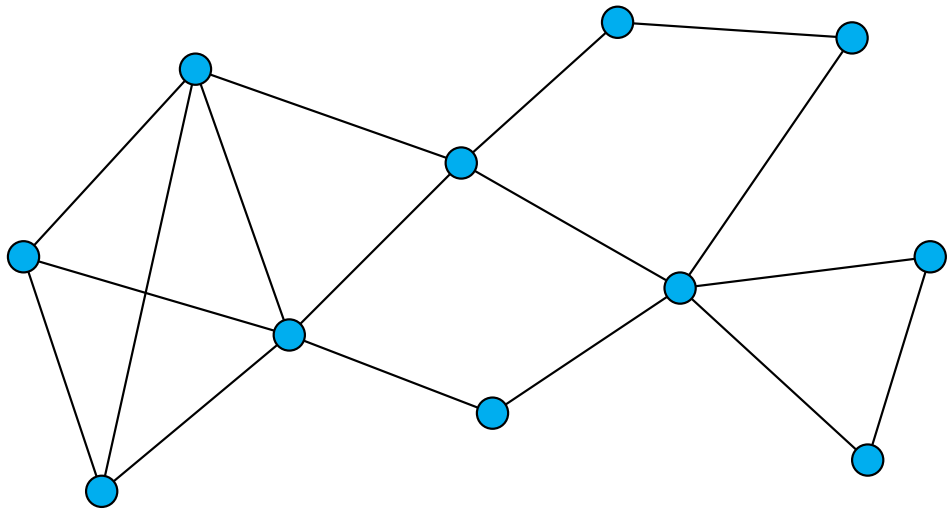
Odpověď: Ano

## CLIQUE – problém kliky

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Existuje v grafu $G$ množina vrcholů velikosti $k$ taková, že každé dva vrcholy této množiny jsou spojeny hranou?

**Příklad:** $k = 4$

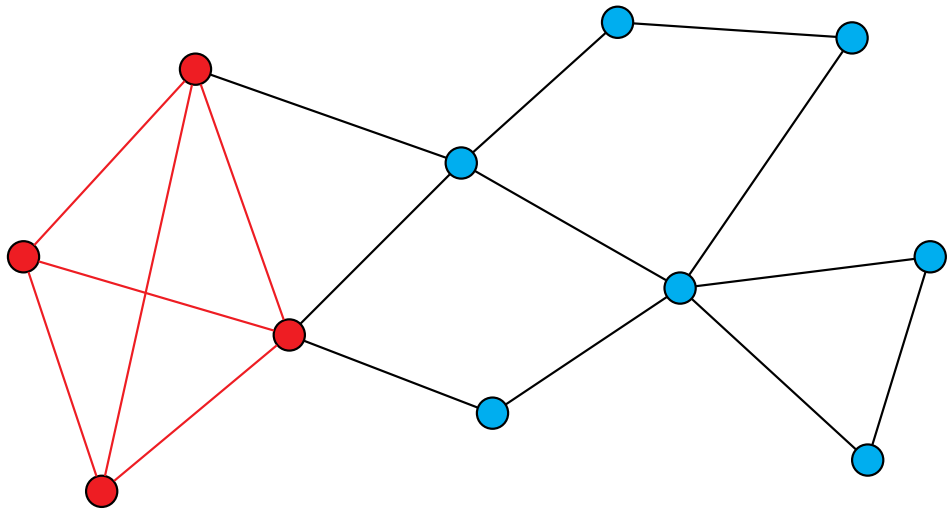


## CLIQUE – problém kliky

Vstup: Neorientovaný graf $G$ a přirozené číslo $k$.

Otázka: Existuje v grafu $G$ množina vrcholů velikosti $k$ taková, že každé dva vrcholy této množiny jsou spojeny hranou?

**Příklad:** $k = 4$



Odpověď: ANO

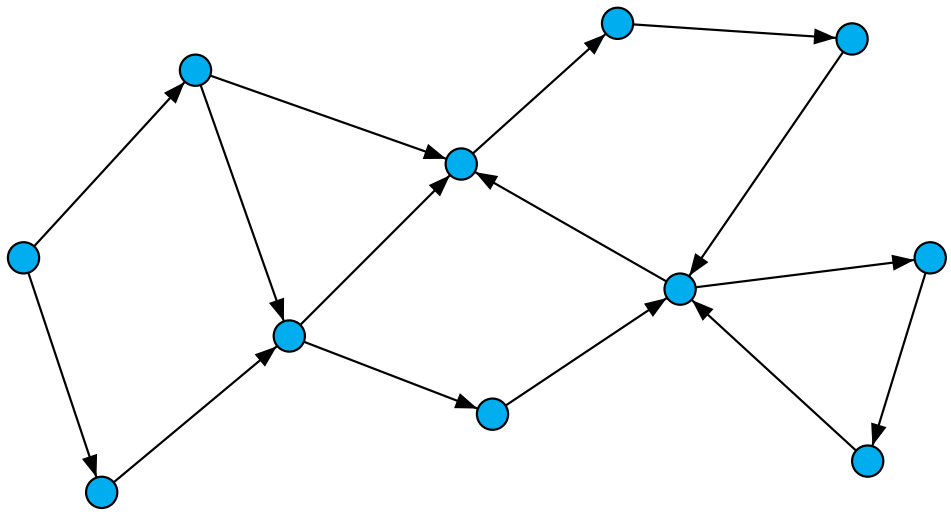
# Hamiltonovský cyklus

## HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf $G$.

Otázka: Existuje v grafu $G$ Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

**Příklad:**



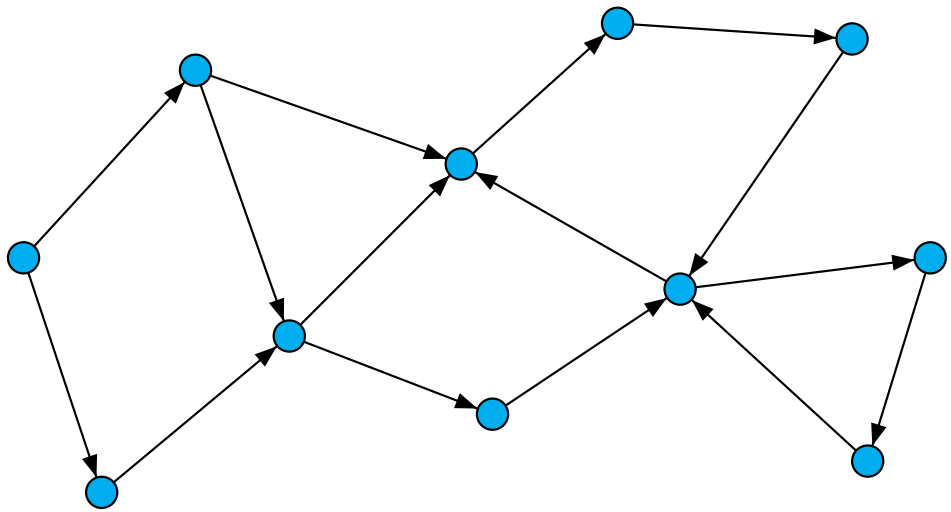
# Hamiltonovský cyklus

## HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf $G$.

Otázka: Existuje v grafu $G$ Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

**Příklad:**



Odpověď: NE

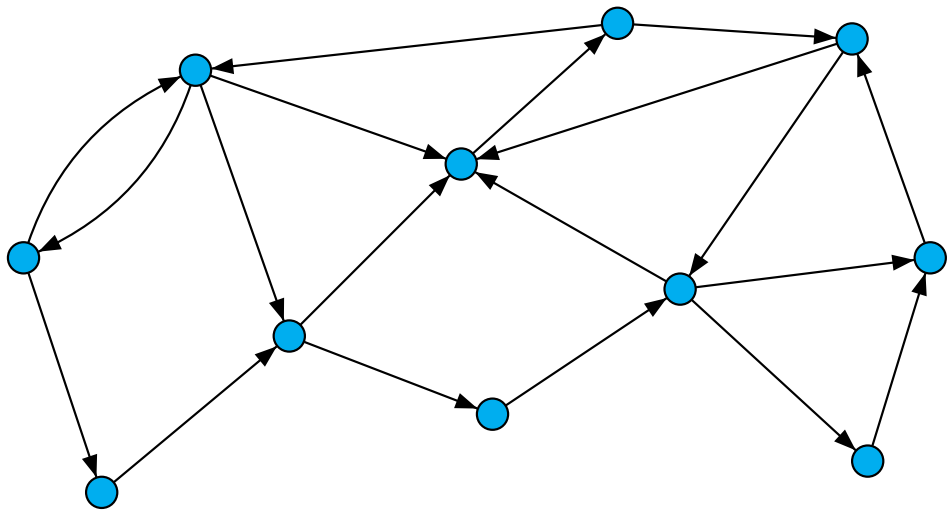
# Hamiltonovský cyklus

## HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf $G$.

Otázka: Existuje v grafu $G$ Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

**Příklad:**



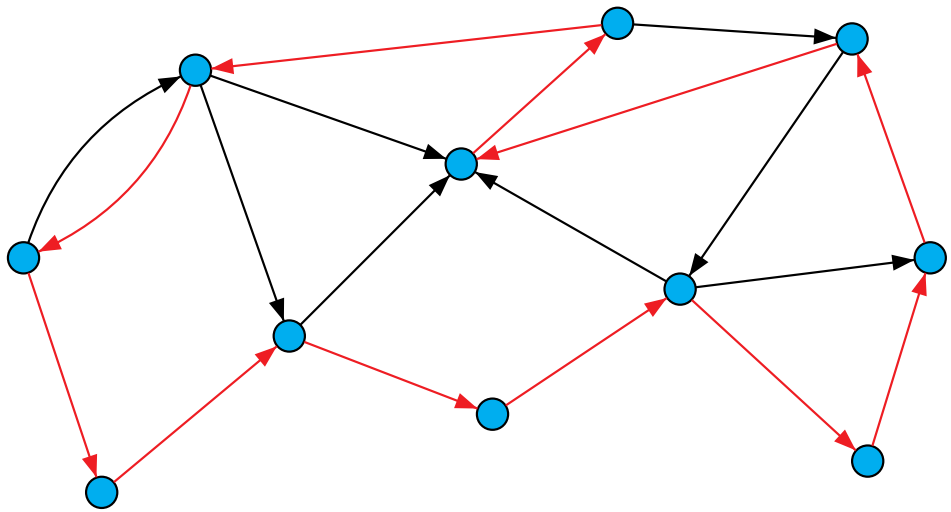
# Hamiltonovský cyklus

## HC – Problém „Hamiltonovský cyklus“

Vstup: Orientovaný graf $G$.

Otázka: Existuje v grafu $G$ Hamiltonovský cyklus (orientovaný cyklus procházející každým vrcholem právě jednou)?

**Příklad:**



Odpověď: Ano

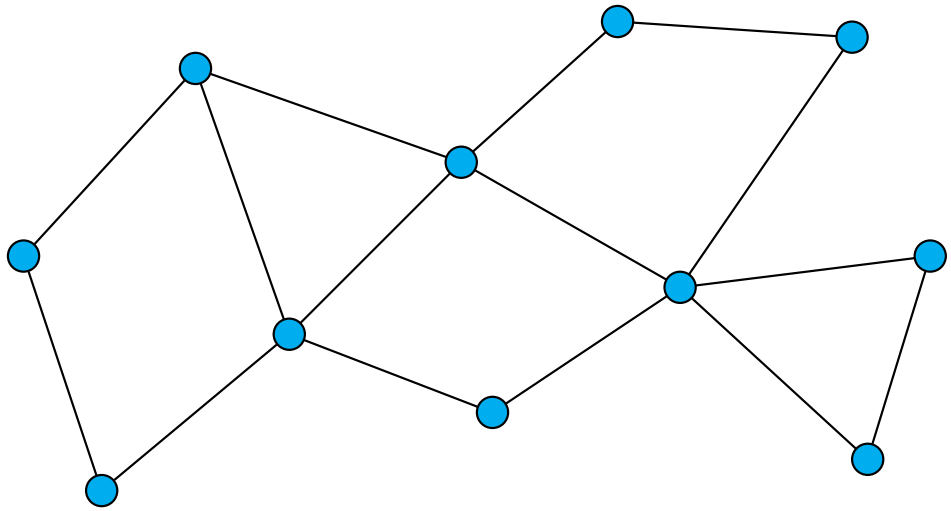
# Hamiltonovská kružnice

## HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf $G$.

Otázka: Existuje v grafu $G$ Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

**Příklad:**



Odpověď: NE

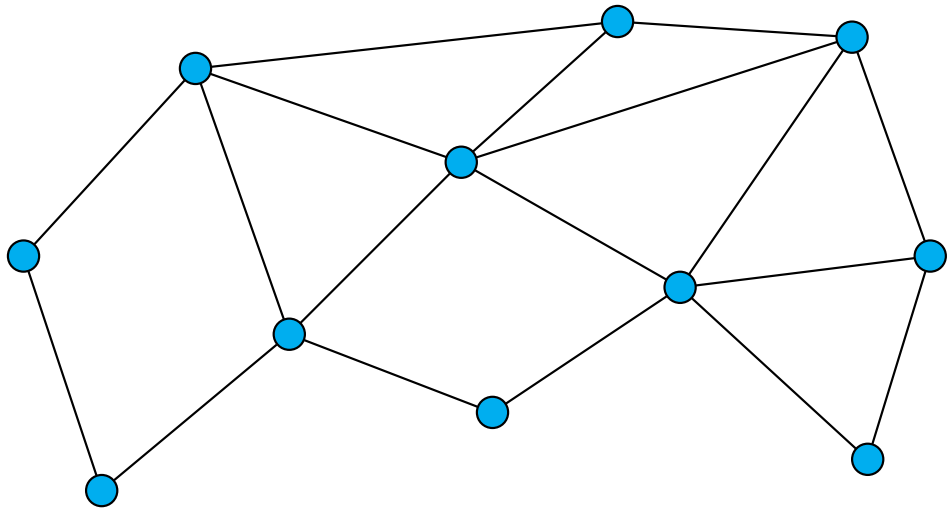
# Hamiltonovská kružnice

## HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf $G$.

Otázka: Existuje v grafu $G$ Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

**Příklad:**



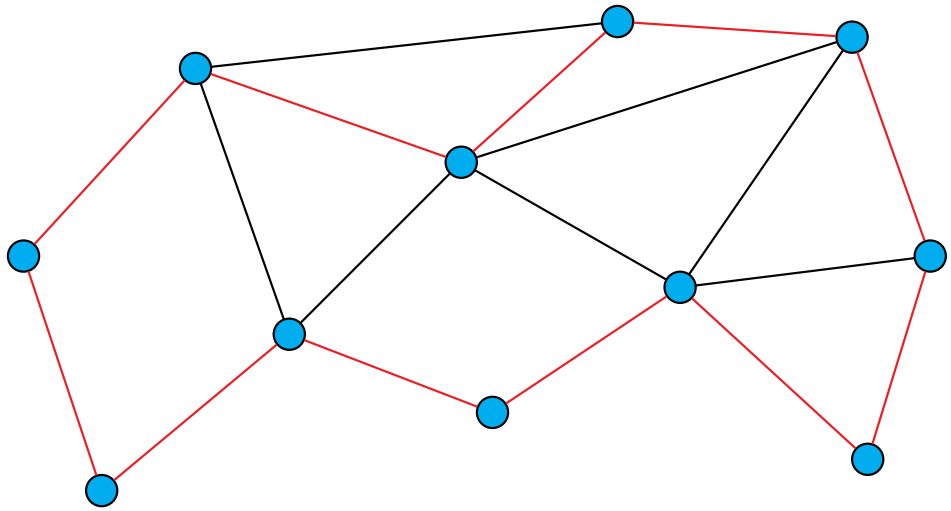
# Hamiltonovská kružnice

## HK – Problém „Hamiltonovská kružnice“

Vstup: Neorientovaný graf $G$.

Otázka: Existuje v grafu $G$ Hamiltonovská kružnice (neorientovaný cyklus procházející každým vrcholem právě jednou)?

**Příklad:**



Odpověď: ANO

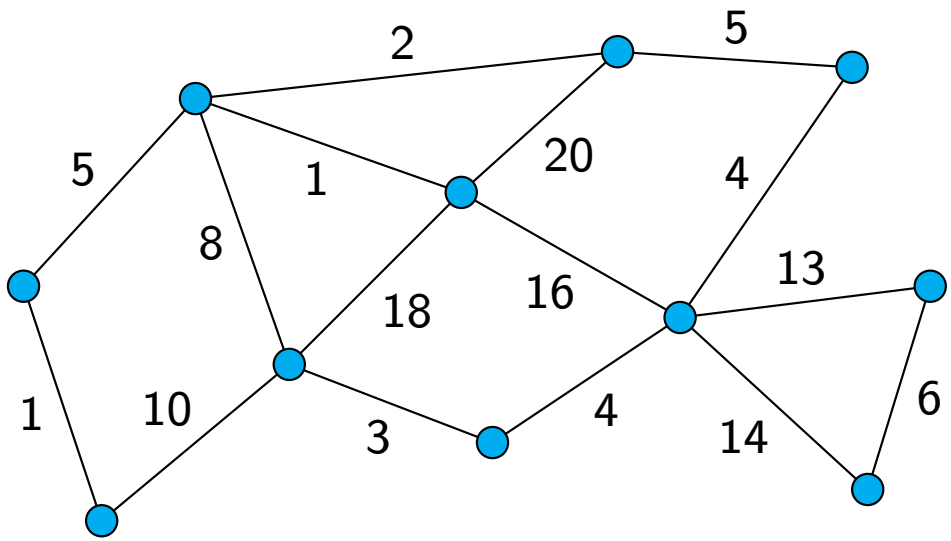
# Problém obchodního cestujícího

## TSP - Problém „obchodního cestujícího“

Vstup: Neorientovaný graf $G$ s hranami ohodnocenými přirozenými čísly a číslo $k$.

Otázka: Existuje v grafu $G$ uzavřený sled procházející všemi vrcholy takový, že součet délek hran na tomto sledu (včetně opakovaných) je maximálně $k$?

**Příklad:** $k = 70$



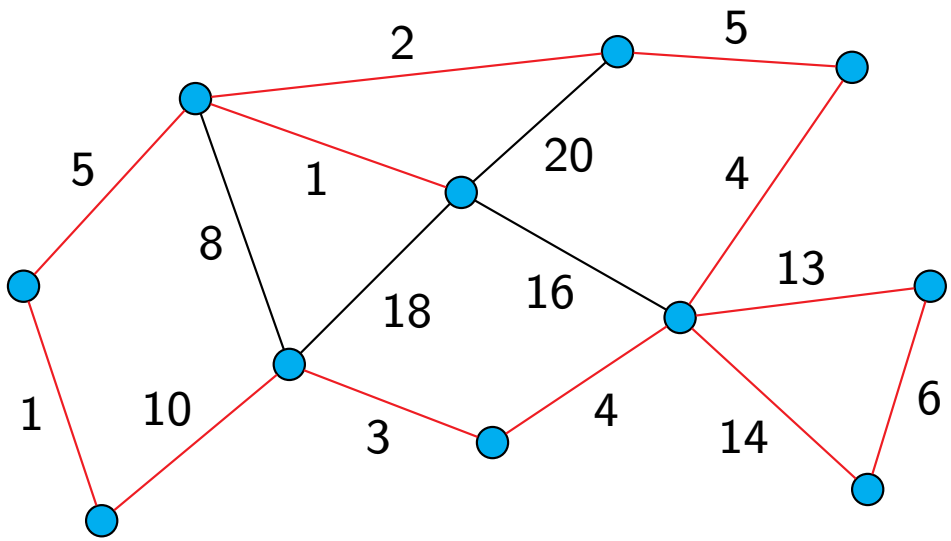
# Problém obchodního cestujícího

## TSP - Problém „obchodního cestujícího“

Vstup: Neorientovaný graf $G$ s hranami ohodnocenými přirozenými čísly a číslo $k$.

Otázka: Existuje v grafu $G$ uzavřený sled procházející všemi vrcholy takový, že součet délek hran na tomto sledu (včetně opakovaných) je maximálně $k$?

**Příklad:** $k = 70$




Odpověď: Ano, protože byl nalezen sled se součtem 69.

# SUBSET-SUM

## Problém SUBSET-SUM

Vstup: Sekvence přirozených čísel $a_1, a_2, \ldots, a_n$ a přirozené číslo $s$.

Otázka: Existuje množina $I \subseteq \{1, 2, \ldots, n\}$ taková, že $\sum_{i \in I} a_i = s$?

Jinak řečeno, ptáme se zda z dané (multi)množiny čísel je možné vybrat podmnožinu, jejíž součet je $s$.

**Příklad:** Pro vstup tvořený čísly $3, 5, 2, 3, 7$ a číslem $s = 15$ je odpověď Ano, neboť $3 + 5 + 7 = 15$.

Pro vstup tvořený čísly $3, 5, 2, 3, 7$ a číslem $s = 16$ je odpověď Ne, neboť žádná podmnožina těchto čísel nedává součet $16$.

# SUBSET-SUM

**Poznámka:**
Pořadí čísel $a_1, a_2, \ldots, a_n$ na vstupu není důležité.

Všimněte si však určitého rozdílu oproti tomu, kdybychom problém formulovali tak, že vstupem je množina $\{a_1, a_2, \ldots, a_n\}$ a číslo $s$:

V množině se čísla neopakují, zatímco v sekvenci se může totéž číslo vyskytnout vícekrát.

# SUBSET-SUM

Problém SUBSET-SUM je speciálním případem **problému batohu** (knapsack problem):

## Knapsack problem

Vstup: Sekvence dvojic přirozených čísel $(a_1, b_1), (a_2, b_2), \ldots, (a_n, b_n)$ a dvě přirozená čísla $s$ a $t$.

Otázka: Existuje množina $I \subseteq \{1, 2, \ldots, n\}$ taková, že $\sum_{i \in I} a_i \leq s$ a $\sum_{i \in I} b_i \geq t$ ?

# SUBSET-SUM

Neformálně můžeme problém batohu formulovat takto:

Máme $n$ předmětů, kde $i$-tý předmět váží $a_i$ gramů a má cenu $b_i$ Kč. Do batohu se vejdou předměty o maximální celkové váze $s$ gramů.

Otázka zní, zda můžeme z předmětů vybrat podmnožinu, která by vážila maximálně $s$ gramů a měla celkovou cenu alespoň $t$ Kč.

**Poznámka:**
Zde jsme problém batohu formulovali jako rozhodovací problém.

Běžnější je formulovat tento problém jako optimalizační problém, kde je cílem najít takovou množinu $I \subseteq \{1, 2, \ldots, n\}$, kde hodnota $\sum_{i \in I} b_i$ je maximální, přičemž ovšem musí být dodržena podmínka $\sum_{i \in I} a_i \leq s$, tj. vybrat předměty s maximální celkovou cenou tak, aby nebyla překročena kapacita batohu.

# SUBSET-SUM

To, že SUBSET-SUM je speciálním případem problému batohu, vidíme z následující (téměř triviální) redukce:

Řekněme, že $a_1, a_2, \ldots, a_n$, $s_1$ je instance problému SUBSET-SUM. Je očividné, že pro instanci problému batohu, kde máme sekvenci $(a_1, a_1), (a_2, a_2), \ldots, (a_n, a_n)$, $s = s_1$ a $t = s_1$, je odpověď stejná jako pro původní instanci SUBSET-SUM.

# SUBSET-SUM

Pokud chceme studovat složitost problémů jako jsou SUBSET-SUM nebo problém batohu, je dobré si nejprve ujasnit, co považujeme za velikost vstupu.

Asi nejpřirozenější je definovat velikost vstupu jako celkový počet bitů, který potřebujeme k zápisu instance.

Musíme však určit, jakým způsobem jsou na vstupu zadána přirozená čísla – zda binárně (případně v jiné číselné soustavě o základu alespoň 2, např. desítkové nebo šestnáctkové) nebo unárně.

Pokud uvažujeme, že čísla jsou na vstupu zadána **binárně**, tj. že velikost vstupu je úměrná součtu délek binárních zápisů jednotlivých čísel na vstupu, tak problém SUBSET-SUM je NP-těžký.
(Dá se ukázat polynomiální převod z 3-SAT.)

# SUBSET-SUM

Není těžké se přesvědčit, že SUBSET-SUM i problém batohu (jeho rozhodovací varianta) patří do třídy NPTIME:

- Nedeterministický algoritmus nejprve nedeterministicky zvolí podmnožinu prvků sekvence na vstupu a poté (deterministicky) ověří, zda splňuje splňuje danou podmínku (resp. podmínky).

  Je zřejmé, že toto ověření je možné provést v čase polynomiálním vzhledem k velikosti instance.

Problémy SUBSET-SUM i problém batohu jsou tedy NP-úplné.

# ILP – celočíselné lineární programování

## Problém ILP (celočíselné lineární programování)

Vstup: Celočíselná matice $A$ a celočíselný vektor $b$.

Otázka: Existuje celočíselný vektor $x$, takový že $Ax \leq b$?

Příklad instance problému:

$$A = \begin{pmatrix} 3 & -2 & 5 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix} \qquad b = \begin{pmatrix} 8 \\ -3 \\ 5 \end{pmatrix}$$

Ptáme se tedy, zda existuje celočíselné řešení následující soustavy nerovnic:

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

# ILP – celočíselné lineární programování

Jedním z řešení soustavy

$$\begin{aligned} 3x_1 - 2x_2 + 5x_3 &\leq 8 \\ x_1 + x_3 &\leq -3 \\ 2x_1 + x_2 &\leq 5 \end{aligned}$$

je například $x_1 = -4$, $x_2 = 1$, $x_3 = 1$, tj.

$$x = \begin{pmatrix} -4 \\ 1 \\ 1 \end{pmatrix}$$

neboť

$$\begin{aligned} 3 \cdot (-4) - 2 \cdot 1 + 5 \cdot 1 &= -9 \leq 8 \\ -4 + 1 &= -3 \leq -3 \\ 2 \cdot (-4) + 1 &= -7 \leq 5 \end{aligned}$$

Pro tuto instanci je tedy odpověď Ano.

# ILP – celočíselné lineární programování

**Poznámka:** Analogický problém, kdy se pro danou soustavu lineálních nerovnic ptáme, zda existuje její řešení v oboru **reálných** čísel, je možné řešit v polynomiálním čase.

# P vs. NP

Jak už bylo zmíněno, otázka, zda $\mathrm{PTIME} = \mathrm{NPTIME}$, je dlouhodobě otevřený problém.

Obecně se má za to, že pravděpodobně $\mathrm{PTIME} \neq \mathrm{NPTIME}$, dosud to však nikdo nedokázal.

**Poznámka:** Například se dá ukázat, že $\mathrm{PTIME} \neq \mathrm{NPTIME}$ je nutnou (nikoli však postačující) podmínkou pro to, aby vůbec mohly existovat šifrovací algoritmy, které by nebyly snadno prolomitelné.

Pokud by se podařilo někomu najít polynomiální algoritmus pro alespoň jeden NP-úplný problém, okamžitě bychom tím získali algoritmy pro rychlé prolomení všech v současné době používaných šifer.

# Jak řešit těžké problémy?

Pokud pro daný problém neexistuje efektivní algoritmus, máme následující možnosti:

1. **Exponenciální algoritmy** – použitelné jen na malé instance.
2. **Aproximační algoritmy** – použitelné jen pro optimalizační problémy. Najde řešení, které je o něco horší než optimum.
3. **Pravděpodobnostní (randomizované) algoritmy** – najde rychle řešení, ale řešení je s určitou pravděpodobností špatně.
4. **Speciální případy** – soustředit se jen na některé speciální případy instancí, neřešit problém v plné obecnosti.
5. **Heuristiky** – postup, který najde řešení rychle ve většině „rozumných“ případů, není však zaručeno, že vždy.

# Nerozhodnutelné problémy

# Algoritmicky nerozhodnutelné problémy

Problém, který není algoritmicky řešitelný je **algoritmicky neřešitelný**.

Rozhodovací problém, který není rozhodnutelný je **nerozhodnutelný**.

**Příklad:** Následující problém zvaný **Problém zastavení (Halting problem)** je nerozhodnutelný:

## Halting problem

Vstup: Popis Turingova stroje $M$ a slovo $w$.

Otázka: Zastaví se stroj $M$ po nějakém konečném počtu kroků, pokud dostane jako svůj vstup slovo $w$?

# Halting problem

Alternativně bychom ho mohli formulovat třeba takto:

## Halting problem

Vstup: Zdrojový kód programu $P$ v jazyce C, vstupní data $x$.

Otázka: Zastaví se program $P$ po nějakém konečném počtu kroků, pokud dostane jako vstup data $x$?

# Halting problem

Předpokládejme, že by existoval nějaký program, který by rozhodoval Halting problem.

Mohli bychom tedy vytvořit podprogram $H$, deklarovaný jako

boolean H(String kod, String vstup)

kde $H(P, x)$ vrátí:

- true pokud se program $P$ zastaví pro vstup $x$,
- false pokud se program $P$ nezastaví pro vstup $x$.

**Poznámka:** Řekněme, že podprogram $H(P, x)$ by vracel false v případě, že $P$ není syntakticky správný kód programu.

# Halting problem

S použitím podprogramu $H$ bychom vytvořili program $D$, který bude provádět následující kroky:

- Načte svůj vstup do proměnné $x$ typu String.
- Zavolá podprogram $H(x, x)$.
- Pokud podprogram $H$ vrátil true, skočí do nekonečné smyčky

```
loop: goto loop
```

V případě, že $H$ vrátil false, program $D$ se ukončí.

Co udělá program $D$, pokud mu předložíme jako vstup jeho vlastní kód?

# Halting problem

Pokud $D$ dostane jako vstup svůj vlastní kód, tak se buď zastaví nebo nezastaví.

- Pokud se $D$ zastaví, tak $H(D, D)$ vrátí true a $D$ skočí do nekonečné smyčky. Spor!
- Pokud se $D$ nezastaví, tak $H(D, D)$ vrátí false a $D$ se zastaví. Spor!

V obou případech dospějeme ke sporu a další možnost není. Nemůže tedy platit předpoklad, že $H$ řeší Halting problem.

# Další nerozhodnutelné problémy

S jedním příkladem nerozhodnutelného problému už jsme se setkali:

**Problém**

Vstup: Bezkontextové gramatiky $G_1$ a $G_2$.

Otázka: Je $L(G_1) = L(G_2)$?

případně

**Problém**

Vstup: Bezkontextová gramatika $G$ generující jazyk nad abecedou $\Sigma$.

Otázka: Je $L(G) = \Sigma^*$?

# Redukce mezi problémy

Pokud máme o nějakém (rozhodovacím) problému dokázáno, že je nerozhodnutelný, můžeme ukázat nerozhodnutelnost dalších problémů pomocí redukcí.

Řekněme, že $A$ a $B$ jsou rozhodovací problémy.
**Redukce** problému $A$ na problém $B$ je algoritmus $P$, který:

- Dostane jako vstup instanci problému $A$ (označme ji $x$).
- Jako svůj výstup (označme jej $P(x)$) vyprodukuje instanci problému $B$.
- Pro každou instanci $x$ problému $A$ platí, že pro vstup $x$ je v problému $A$ odpověď ANO právě tehdy, když pro vstup $P(x)$ je v problému $B$ odpověď ANO.

# Redukce mezi problémy

Řekněme, že existuje redukce $P$ problému $A$ na problém $B$.

Pokud by problém $B$ byl rozhodnutelný, pak i problém $A$ je rozhodnutelný.

Řešení problému $A$ pro vstup $x$:

- Zavoláme $P$ se vstupem $x$, vrátí nám hodnotu $P(x)$.
- Zavoláme algoritmus řešící problém $B$ se vstupem $P(x)$.
- Hodnotu, kterou nám vrátí vypíšeme jako výsledek.

Je zřejmé, že pokud $A$ je nerozhodnutelný, tak $B$ nemůže být rozhodnutelný.

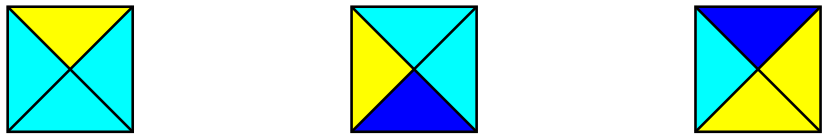
# Další nerozhodnutelné problémy

Redukcí z Halting problému se dá ukázat nerozhodnutelnost celé řady problémů, které se týkají ověřování chování programů:

- Vydá daný program pro nějaký vstup odpověď ANO?
- Zastaví se daný program pro libovolný vstup?
- Dávají dva dané programy pro stejné vstupy stejný výstup?
- ...

# Další nerozhodnutelné problémy

Vstupem je množina typů kachliček, jako třeba:

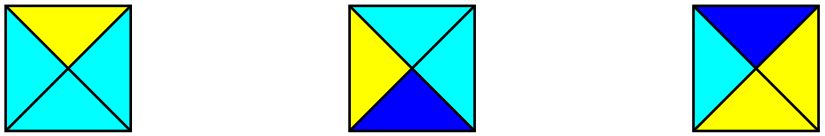


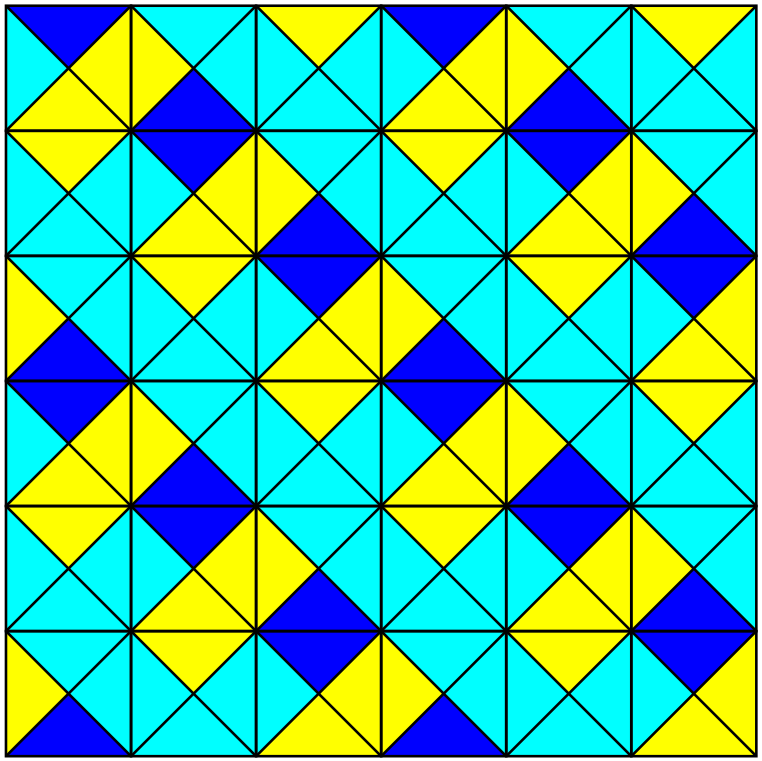
Otázka je, zda je možné použitím daných typů kachliček pokrýt každou libovolně velkou konečnou plochu tak, aby všechny kachličky spolu sousedily stejnými barvami.

**Poznámka:** Můžeme předpokládat, že máme v zásobě neomezené množství kachliček všech typů.

Kachličky není dovoleno otáčet.

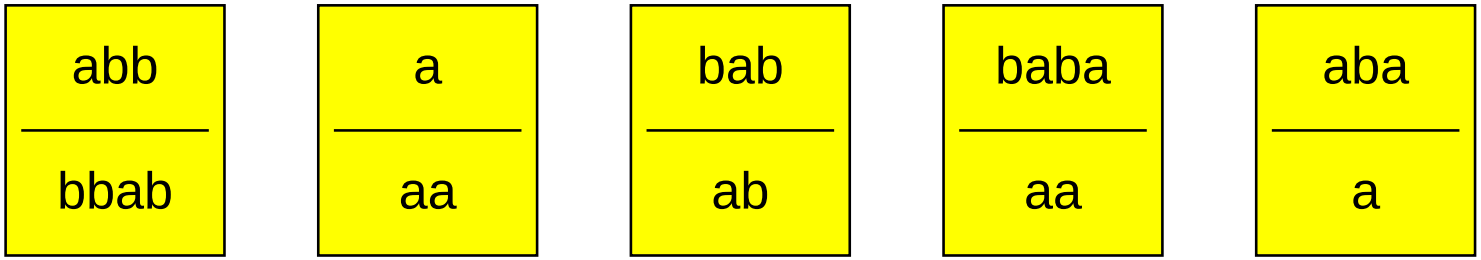
# Další nerozhodnutelné problémy



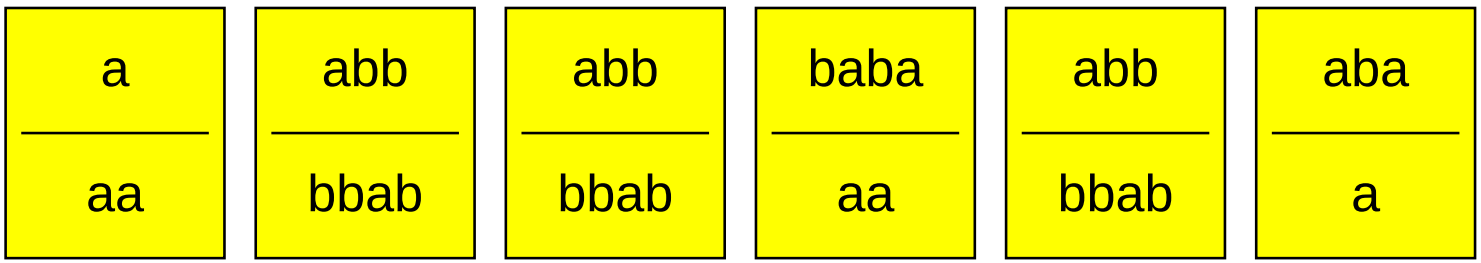


# Další nerozhodnutelné problémy

Vstupem je množina typů kartiček, jako třeba:




Otázka je, zda je možné z těchto typů kartiček vytvořit neprázdnou konečnou posloupnost, kde zřetězením slov nahoře i dole vznikne totéž slovo. Každý typ kartičky je možné používat opakovaně.




Nahoře i dole vznikne slovo aabbabbbabaabbaba.

# Další nerozhodnutelné problémy

Redukcí z předchozího problému se dá snadno ukázat nerozhodnutelnost některých dalších problémů z oblasti bezkontextových gramatik:

**Problém**

Vstup: Bezkontextové gramatiky $G_1$ a $G_2$.

Otázka: Je $L(G_1) \cap L(G_2) = \emptyset$?

**Problém**

Vstup: Bezkontextová gramatika $G$.

Otázka: Je $G$ nejednoznačná?

# Další nerozhodnutelné problémy

## Problém

Vstup: Uzavřená formule PL1, ve které mohou být použity jako funkční symboly $+$ a $*$ a celočíselné konstanty a jako predikátové symboly $=$ a $<$.

Otázka: Je daná formule pravdivá v oboru přirozených čísel (při přirozené interpretaci všech funkčních a predikátových symbolů)?

Příklad vstupu:

$$\forall x \exists y \forall z((x * y = z) \wedge (y + 5 = x))$$

**Poznámka:** Úzce souvisí s Gödelovou větou o neúplnosti.

# Další nerozhodnutelné problémy

Je zajímavé, že analogický problém, kde ale místo přirozených čísel uvažujeme čísla reálná, je algoritmicky rozhodnutelný (i když popis daného algoritmu a důkaz jeho korektnosti jsou značně netriviální).

Rovněž pokud uvažujeme přirozená nebo celá čísla a stejné formule jako v předchozím případě, ale s tím, že v nich nesmí být použit funkční symbol $*$ (násobení), tak je problém algoritmicky rozhodnutelný.

# Další nerozhodnutelné problémy

Pokud můžeme používat $*$, je ve skutečnosti je nerozhodnutelný už velmi omezený případ:

## Desátý Hilbertův problém

**Vstup:** Polynom $f(x_1, x_2, \ldots, x_n)$ vytvořený z proměnných $x_1, x_2, \ldots, x_n$ a celočíselných konstant.

**Otázka:** Existují přirozená čísla $x_1, x_2, \ldots, x_n$ taková, že $f(x_1, x_2, \ldots, x_n) = 0$?

Příklad vstupu: $5x^2y - 8yz + 3z^2 - 15$

Tj. ptáme se, zda

$$\exists x \exists y \exists z (5 * x * x * y + (-8) * y * z + 3 * z * z + (-15) = 0)$$

platí v oboru přirozených čísel.

# Další nerozhodnutelné problémy

Také následující problém je algoritmicky nerozhodnutelný:

**Problém**

Vstup: Uzavřená formule $\varphi$ PL1.

Otázka: Platí $\models \varphi$ ?

**Poznámka:** Zápis $\models \varphi$ znamená, že formule $\varphi$ je logicky platná, tj. pravdivá v každé interpretaci.

# Částečně rozhodnutelné problémy

Problém je **částečně rozhodnutelný**, jestliže existuje algoritmus, který:

- Pokud dostane jako vstup instanci, pro kterou je odpověď ANO, tak se po konečném počtu kroků zastaví a vypíše `"ANO"`.
- Pokud dostane jako vstup instanci, pro kterou je odpověď NE, tak se buď zastaví a vypíše `"NE"` nebo se nikdy nezastaví.

Je očividné, že například HP (Halting problem) je částečně rozhodnutelný.

Některé problémy však nejsou ani částečně rozhodnutelné.