# Řešení výpočetně náročných problémů

# Řešení výpočetně náročných problémů

- Pro mnoho důležitých problémů nejsou známy efektivní algoritmy. Řada těchto problémů je například NP-úplných.
- Je možné, že pro tyto problémy ani polynomiální algoritmy neexistují a my to zatím jenom neumíme dokázat.
- Přesto potřebujeme tyto problémy řešit.

# Řešení výpočetně náročných problémů

Pokud chceme řešit nějaký problém, tak v ideálním případě chceme použít k jeho řešení algoritmus, který:

- Bude **polynomiální**, tj. rychle skončí pro libovolnou vstupní instanci.
- Bude **korektní**, tj. pro libovolnou vstupní instanci najde správnou odpověď.

Pokud nevíme, jak takový algoritmus najít, musíme trochu slevit ze svých požadavků.

# Řešení výpočetně náročných problémů

- **Obtížné instance versus typické instance**

  Při studiu složitosti problémů zkoumáme většinou složitost v nejhorším případě.

  Instance, kvůli kterým je problém těžký, mohou být dost odlišné od „typických“ instancí, pro které chceme problém řešit.

  Při podrobnějším zkoumání problému můžeme nalézt určitou podmnožinu instancí, pro které je možné problém rychle řešit.

## Problém batohu

Vstup: Čísla $a_1, a_2, \ldots, a_m$ a číslo $s$.

Otázka: Existuje podmnožina množiny čísel $a_1, a_2, \ldots, a_m$ taková, že součet čísel v této podmnožině je $s$?

**Poznámka:** Jako velikost instance bereme celkový počet bitů v zápisu čísel $a_1, a_2, \ldots, a_m$ a $s$.

Tento problém je NP-úplný. Neumíme ho v rozumném čase řešit, pokud čísla v instanci budou „velká“ (např. 1000-bitová).

Pokud je však hodnota $s$ malá (např. do 1000000, tj. asi 20 bitů), je možné tento problém vyřešit během několika sekund i pro relativně velké hodnoty $m$ (např. $m = 10000$).

## Problém „HORN-SAT“

Vstup: Booleovská formule $\varphi$ v KNF obsahující pouze Hornovy klauzule.

Otázka: Je $\varphi$ splnitelná?

Problém HORN-SAT se dá (narozdíl od obecného problému SAT) řešit v polynomiálním čase.

**Poznámka:** Hornova klauzule je klauzule, ve které se nachází nejvýše jeden pozitivní literál.

Například $(\neg x_1 \vee \neg x_2 \vee \neg x_3 \vee \neg x_4 \vee x_5)$ je Hornova klauzule.

Všimněte si, že tato klauzule je ekvivalentní formuli

$$(x_1 \wedge x_2 \wedge x_3 \wedge x_4) \Longrightarrow x_5$$

# Řešení výpočetně náročných problémů

Následující problém je rovněž možné řešit v polynomiálním čase:

**Problém „2-SAT“**

Vstup: Booleovská formule $\varphi$ v KNF, kde každá klauzule obsahuje nejvýše 2 literály.

Otázka: Je $\varphi$ splnitelná?

**Příklad:**

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

**Exponenciální algoritmy**

I exponenciální algoritmus může být prakticky použitelný, pokud:

- Instance, pro které problém chceme řešit, jsou poměrně malé.
- Složitost je sice exponenciální, ale roste pomaleji než $2^n$.
- Algoritmus je co nejoptimálněji naprogramován.

**Příklad:**

- $f(n) = (1.2)^n$ pro $n = 100$ je $f(n) \approx 86 \cdot 10^6$.
- $f(n) = 10 \cdot 2^{\sqrt{n}}$ pro $n = 300$ je $f(n) \approx 1.64 \cdot 10^6$.

# Řešení výpočetně náročných problémů

## Menší požadavky na korektnost

- **Randomizované algoritmy** – algoritmy, které využívají při výpočtu generátor náhodných čísel.

  Existuje určitá nenulová pravděpodobnost, že algoritmus vrátí chybný výsledek.

  Pro libovolně malé $\varepsilon > 0$ jsme však schopni zaručit, že pravděpodobnost chyby není větší než $\varepsilon$.

- **Aproximační algoritmy** – používají se pro řešení optimalizačních problémů.

  U těchto algoritmů není zaručeno, že naleznou optimální řešení, ale je zaručeno, že naleznou řešení, které nebude o moc horší než optimální řešení (např. nanejvýš $2\times$ horší).

# Randomizované algoritmy

# Testování prvočíselnosti

## Prvočíselnost

Vstup: Přirozené číslo $p$.

Otázka: Je $p$ prvočíslo?

Pro „malá“ $p$ (např. do $10^{12}$) stačí vyzkoušet čísla od 2 do $\lfloor \sqrt{p} \rfloor$, zda některé z nich dělí $p$.

Tento problém má velký význam v kryptografii, kde ho potřebujeme řešit pro čísla, která mají délku několik tisíc bitů.

# Testování prvočíselnosti

To, zda je možné testovat prvočíselnost v polynomiálním čase, byl dlouho otevřený problém.

Teprve od roku 2003 je znám polynomiální algoritmus. Původní verze algoritmu měla složitost $\mathcal{O}(n^{12+\varepsilon})$, později o něco zlepšen ($\mathcal{O}(n^{7.5})$). Aktuálně má nejrychlejší známý algoritmus složitost $\mathcal{O}(n^6)$.

V praxi se používají **randomizované** algoritmy se složitostí $\mathcal{O}(n^3)$:

- Solovay-Strassen
- Miller-Rabin

**Poznámka:** $n$ zde označuje počet bitů čísla $p$.

# Testování prvočíselnosti

S problémem testování prvočíselnosti úzce souvisí následující problém.

## Faktorizace

Vstup: Přirozené číslo $p$.

Výstup: Rozklad čísla $p$ na prvočísla.

Na rozdíl od prvočíselnosti není pro problém faktorizace znám žádný efektivní algoritmus, a to ani randomizovaný.

Jedna z používaných šifer je tzv. RSA kryptosystém, který je založen na tom, že:

- Umíme rychle najít velká prvočísla (a rychle je mezi sebou vynásobit).
- Není známo, jak v rozumném čase z tohoto součinu zjistit původní prvočísla.

# Testování prvočíselnosti

**Poznámka:** Pokud umíme rychle testovat, zda je dané číslo prvočíslem, není problém velké prvočíslo náhodně vygenerovat, neboť je známo, že prvočísla jsou mezi ostatními čísly rozmístěna poměrně „hustě“.

$\pi(n)$ – počet prvočísel v intervalu $1, 2, \ldots, n$

Je dokázáno, že

$$\lim_{n \to \infty} \frac{\pi(n)}{n / \ln n} = 1$$

neboli jinak řečeno, mezi čísly od $1$ do $n$ je zhruba $n / \ln n$ prvočísel.

Pokud tedy chceme náhodně vygenerovat $k$-bitové prvočíslo, stačí zhruba $k$ pokusů.

# Randomizované algoritmy

Randomizovaný algoritmus:

- používá během výpočtu generátor náhodných čísel
- může pro tentýž vstup skončit s různým výsledkem
- existuje určitá pravděpodobnost, že vrátí chybný výsledek

Aby měl randomizovaný algoritmus praktický smysl, musíme mít možnost regulovat pravděpodobnost chyby:

> *Pro libovolně malé* $\varepsilon > 0$ *musíme být schopni zaručit, že pravděpodobnost chyby nebude větší než* $\varepsilon$.

# Randomizované algoritmy

Vezměme si například algoritmus Miller-Rabin pro testování prvočíselnosti.

Pro libovolné $n$ vrací buď odpověď „`Prvočíslo`“ nebo odpověď „`Složené`“.

- Pokud $n$ je prvočíslo, vrátí vždy odpověď „`Prvočíslo`“.
- Pokud $n$ je číslo složené, je pravděpodobnost toho, že vrátí odpověď „`Složené`“ nejméně $50\%$.

  Může však vrátit (chybnou) odpověď „`Prvočíslo`“.

Algoritmus můžeme spouštět opakovaně. Výsledek při dalším spuštění je nezávislý na výsledcích z předchozích spuštění.

# Randomizované algoritmy

Řekněme, že algoritmus spustíme $m$ krát (pro tentýž vstup $n$).

- Pokud alespoň jednou dostaneme odpověď „`Složené`“, pak víme s jistotou, že $n$ je číslo složené.
- Pokud pokaždé dostaneme odpověď „`Prvočíslo`“, pak pravděpodobnost toho, že $n$ není prvočíslo je maximálně

$$\left(\frac{1}{2}\right)^m = 2^{-m}$$

Například pro $m = 100$ je pravděpodobnost chyby zanedbatelně malá.

# Randomizované algoritmy

**Poznámka:** Lze například odvodit, že pravděpodobnost toho, že počítač bude zasažen během dané mikrosekundy meteoritem, je nejméně $2^{-100}$ za předpokladu, že každých 1000 let je meteoritem zničeno alespoň $100\,\mathrm{m}^2$ zemského povrchu.

# Randomizované algoritmy

Randomizované algoritmy jsou typicky založeny na hledání **svědků** (witness), kteří „dosvědčí“ určitou odpověď vydanou algoritmem.

Tyto svědky vybíráme z množiny **potenciálních svědků**:

- Náhodně vybereme nějakého potenciálního svědka.
- Ověříme, zda se jedná o skutečného svědka, a podle toho vydáme odpověď.

Například v algoritmu Miller-Rabin hledáme svědky složenosti čísla $n$.

Vybíráme je z množiny čísel $\{2, 3, \ldots, n-1\}$:

- Pokud $n$ je prvočíslo, žádní svědci složenosti neexistují.
- Pokud $n$ není prvočíslo, je zaručeno, že alespoň polovina čísel v množině $\{2, 3, \ldots, n-1\}$ jsou svědci složenosti $n$.

# Testování prvočíselnosti

## Malá Fermatova věta:

Pokud $n$ je prvočíslo, pak pro každé $a$ z množiny $\{1, 2, \ldots, n-1\}$ platí

$$a^{n-1} \equiv 1 \pmod{n}$$

Malou Fermatovu větu lze použít k testování prvočíselnosti (tzv. Fermatův test):

- Pro dané $n$ zvolíme nějaké $a \in \{2, 3, \ldots, n-1\}$.
- Pokud $a^{n-1} \not\equiv 1 \pmod{n}$, pak $n$ určitě není prvočíslo.
- Pokud $a^{n-1} \equiv 1 \pmod{n}$, pak $n$ je možná prvočíslo.

# Testování prvočíselnosti

Je překvapivé, že tato procedura vrací chybný výsledek jen poměrně zřídka:

- Pro $a = 2$ a $n < 10000$ je jen $22$ hodnot, pro které dostaneme chybnou odpověď.
- Pro $a = 2$ a $n \to \infty$ se pravděpodobnost toho, že pro náhodně vybrané $n$ dostaneme chybnou odpověď, blíží nule.

# Testování prvočíselnosti

Fermatův test není 100% spolehlivý. Existují složená čísla, která splňují podmínku $a^{n-1} \equiv 1 \pmod{n}$ pro všechna $a \in \{1, 2, \ldots, n-1\}$ nesoudělná s $n$.

Tato čísla se nazývají **Carmichaelova čísla** a jsou extrémně vzácná.

Například mezi čísly do $10^8$ existuje jen 255 Carmichaelových čísel.

Prvních 15 Carmichaelových čísel:

561, 1105, 1729, 2465, 2821, 6601, 8911, 10585, 15841, 29341, 41041, 46657, 52633, 62745, 63973

# Umocňování v modulární aritmetice

## Problém

Vstup: Přirozená čísla $a, b, n$.

Výstup: Hodnota $a^b \bmod n$.

Využijeme toho, že $a^{2i} = a^i \cdot a^i$ a $a^{2i+1} = a^i \cdot a^i \cdot a$.

---

MODULAR-EXPONENTIATION $(a, b, n)$:

```
d := 1
// ⟨b_k, b_{k−1}, ..., b_0⟩ je binární reprezentace b
for i := k downto 0 do
    d := (d · d) mod n
    if b_i = 1 then  d := (d · a) mod n
return d
```

---

# Testování prvočíselnosti (Miller-Rabin)

Na umocňování je založen i následující algoritmus pro testování prvočíselnosti:

```
MILLER-RABIN(n, s):
    for j := 1 to s do
        a := RANDOM(2, n − 1)
        if WITNESS(a, n) then
            return "Složené"   // n je zcela jistě složené
    return "Prvočíslo"   // n je s velkou pravděpodobností prvočíslo
```

Využívá malou Fermatovu větu a toho, že platí následující tvrzení:

Jestliže $p$ je liché prvočíslo, pak rovnice $x^2 \equiv 1 \pmod{p}$ má právě dvě řešení: $x = 1$ a $x = p - 1$.

# Testování prvočíselnosti (Miller-Rabin)

```
WITNESS (a, n):
    d := 1
    // ⟨b_k, b_{k−1}, ..., b_0⟩ je binární reprezentace n − 1
    for i := k downto 0 do
        x := d
        d := (d · d) mod n
        if d = 1 and x ≠ 1 and x ≠ n − 1 then return TRUE
        if b_i = 1 then d := (d · a) mod n
    if d ≠ 1 then return TRUE
    return FALSE
```

# Randomizované algoritmy

- Množina potenciálních svědků bývá obrovská – většinou exponenciálně velká

  **Poznámka:** Kdyby byla polynomiálně velká mohli bychom systematicky projít všechny potenciální svědky a nepotřebovali bychom randomizovaný algoritmus.

- Musí být zaručeno, že pokud nějací skuteční svědci existují, pak jich je dostatečně mnoho, čímž je zaručeno, že pravděpodobnost, že na nějakého svědka narazíme, je dostatečně vysoká.

# Randomizované algoritmy

Formálně můžeme definovat randomizované algoritmy dvěma různými způsoby.

- Obě tyto varianty jsou vzájemně ekvivalentní.
- U obou jde o to, definovat stroj $\mathcal{M}$ tak, aby byla pro každý vstup $x$ definována **pravděpodobnost** toho, že vstup $x$ bude strojem $\mathcal{M}$ přijat.
- O jaký konkrétní druh stroje se jedná, není příliš podstatné — může to být Turingův stroj, stroj RAM, apod.

# Randomizované algoritmy

První varianta definice stroje $\mathcal{M}$ implementujícího pravděpodobnostní algoritmus:

- Tato varianta je podobná jako v případě **nedeterministických** strojů, ale s tím, že v definici přechodové funkce je při výběru mezi více možnými přechody určeno **pravděpodobnostní rozdělení** na těchto možnostech.
- Uvažujme strom všech možných výpočtů pro daný vstup $x$.
- Pro každou větev výpočtu můžeme spočítat pravděpodobnost toho, že tato větev nastane — stačí vynásobit pravděpodobnosti přechodů na této větvi.
- Celková pravděpodobnost toho, že daný stroj $\mathcal{M}$ přijme daný vstup $x$, je dána jako součet pravděpodobností všech přijímajících výpočtů (tj. pravděpodobnost všech větví, které končí odpovědí ANO).

# Randomizované algoritmy

Druhá varianta definice stroje $\mathcal{M}$ implementujícího pravděpodobnostní algoritmus:

- V této variantě popíšeme **deterministický** stroj $\mathcal{M}$, který má ale dvě vstupní pásky, ze kterých může pouze číst:
  - vstupní pásku, na které je zapsána instance problému $x$

    — tato páska je konečná, hlava se na ní může pohybovat oběma směry
  - jednosměrnou nekonečnou pásku, na které je zapsaná nekonečná náhodná posloupnost symbolů z nějaké abecedy (typicky např. $\{0, 1\}$)

    — hlava se na ní může pohybovat pouze směrem zleva doprava
- Typicky se předpokládá, že všechny obsahy pásky s náhodnými daty jsou stejně pravděpodobné.
- Pravděpodobnost přijetí daného slova $x$ je dána jako střední hodnota pravděpodobnosti toho, že výpočet stroje $\mathcal{M}$ bude přijímající.

# Randomizované algoritmy

Předpokládejme, že máme dán nějaký randomizovaný algoritmus implementovaný strojem $\mathcal{M}$, u kterého je pro každý vstup $x$ určeno, jaká je pravděpodobnost toho, že daný stroj $\mathcal{M}$ přijímá vstup $x$.

(Je jedno, jestli bereme první nebo druhou výše uvedenou variantu definice takového stroje.)

Řekněme, že máme dán nějaký rozhodovací problém $A$.

Definovat, kdy je randomizovaný algoritmus implementovaný strojem $\mathcal{M}$ řešením problému $A$ je možné několika způsoby:

- Tyto možnosti definice toho, kdy randomizovaný algoritmus řeší problém, nejsou vzájemně ekvivalentní (resp. není dokázáno, že by byly).
- Různým možnostem odpovídají různé třídy složitosti týkající se randomizovaných algoritmů.

# Randomizované algoritmy

Verze první:

- Pro vstupy, kde je správná odpověď ANO, musí stroj $\mathcal{M}$ dávat odpověď ANO s pravděpodobností alespoň $1/2$.
- Pro vstupy, kde je správná odpověď NE, musí stroj $\mathcal{M}$ vždy dávat odpověď NE.

**Poznámka:** Hodnota $1/2$ zde není příliš podstatná, stejně dobře by to mohla být jakákoli jiná konstanta $c$ z intervalu $(0, 1)$
(tj. splňující podmínku $0 < c < 1$).

Třída RP (**randomized polynomial time**) je tvořena právě těmi rozhodovacími problémy, pro které existuje randomizovaný algoritmus výše uvedeného typu, který má polynomiální časovou složitost.

**Poznámka:** Analogicky je možné definovat třídu co-RP tvořenou doplňkovými problémy k problémům ze třídy RP.

# Randomizované algoritmy

Verze druhá:

- Pro vstupy, kde je správná odpověď ANO, musí stroj $\mathcal{M}$ dávat odpověď ANO s pravděpodobností alespoň $2/3$.
- Pro vstupy, kde je správná odpověď NE, musí stroj $\mathcal{M}$ dávat odpověď NE s pravděpodobností alespoň $2/3$.

**Poznámka:** Hodnota $2/3$ zde není příliš podstatná, stejně dobře by to mohla být jakákoli jiná konstanta $c$ z intervalu $(0.5, 1)$
(tj. splňující podmínku $0.5 < c < 1$).

Třída BPP (**bounded-error probabilistic polynomial time**) je tvořena právě těmi rozhodovacími problémy, pro které existuje randomizovaný algoritmus výše uvedeného typu, který má polynomiální časovou složitost.

# Randomizované algoritmy

Verze třetí:

- Stroj vrací tři možné odpovědi:
  - Ano
  - Ne
  - Nevím
- Pokud se stroj $\mathcal{M}$ vrátí odpověď Ano nebo Ne, je tato odpověď vždy správně.
- Pravděpodobnost toho, že vrátí odpověď Nevím, může být nejvýše $1/2$.

Třída ZPP (**zero-error probabilistic polynomial time**) je tvořena právě těmi rozhodovacími problémy, pro které existuje randomizovaný algoritmus výše uvedeného typu, který má polynomiální časovou složitost.

# Randomizované algoritmy

Pokud budeme dostávat odpověď NEVÍM, můžeme výpočet opakovat tak dlouho, dokud nedostaneme odpověď ANO nebo NE:

- Náhodná tak bude doba výpočtu.
- Potenciálně může existovat i nekonečný výpočet — jeho pravděpodobnost je ale nulová.
- S rostoucím počtem kroků se pravděpodobnost toho, že se stroj během daného počtu kroků nezastaví, limitně blíží nule.

# Randomizovaný komunikační protokol

Máme dvojici počítačů $C_1$ a $C_2$, které jsou velmi daleko od sebe (např. jeden v Evropě a druhý v Americe).

Oba počítače obsahují tutéž databázi, která by měla obsahovat přesně tatáž data.

Naším cílem je navrhnout vhodný komunikační protokol, pomocí kterého ověříme, že obě databáze obsahují přesně tatáž data.

Označme $n$ počet bitů dat v databázi. Řekněme, že $n = 10^{16}$, a že tedy není možné přenášet celý obsah databáze, nebo by to bylo příliš časově náročné.

**Poznámka:** Není těžké ukázat, že každý deterministický protokol řešící tento problém musí přenést minimálně $n$ bitů.

# Randomizovaný komunikační protokol

**Počáteční situace:**

$C_1$ má $n$-bitovou sekvenci $x = x_1 \cdots x_n$,
$C_2$ má $n$-bitovou sekvenci $y = y_1 \cdots y_n$.

**Cíl:** Zjistit, zda $x = y$.

1. $C_1$ zvolí náhodně prvočíslo $p$ z množiny $\{2, 3, \ldots, n^2\}$.
2. $C_1$ spočte číslo $s = Num(x) \bmod p$ a pošle $C_2$ dvojici $(s, p)$.
3. $C_2$ přijme $(s, p)$ a spočte číslo $t = Num(y) \bmod p$.
   Pokud $s \neq t$, vydá odpověď „$x \neq y$“, jinak vydá odpověď „$x = y$“.

**Poznámka:** $Num(x)$ zde označuje číslo, jehož zápisem ve dvojkové soustavě je sekvence bitů $x$.

# Randomizovaný komunikační protokol

**Objem přenášených dat:** $s$ i $p$ mají maximálně $2 \cdot \lceil \log_2 n \rceil$ bitů, takže se celkem přenáší maximálně $4 \cdot \lceil \log_2 n \rceil$ bitů.

**Příklad:** Pro $n = 10^{16}$ se bude přenášet maximálně $4 \cdot 16 \cdot \lceil \log_2 10 \rceil = 256$ bitů.

**Pravděpodobnost chyby:**

- Pokud $x = y$, pak pro libovolné $p$ platí

$$Num(x) \bmod p = Num(y) \bmod p$$

takže dostaneme vždy odpověď „$x = y$“ (pravděpodobnost chyby je $0$).

# Randomizovaný komunikační protokol

- Pokud $x \neq y$ a dostaneme chybnou odpověď „$x = y$“, znamená to, že

$$z = Num(x) \bmod p = Num(y) \bmod p$$

neboli existují čísla $x', y'$ taková, že

$$Num(x) = x' \cdot p + z \qquad Num(y) = y' \cdot p + z$$

takže $p$ je dělitelem čísla $w = |Num(x) - Num(y)|$, neboť

$$Num(x) - Num(y) = x' \cdot p - y' \cdot p = (x' - y') \cdot p$$

Prvočíslo $p$ bylo vybíráno náhodně z množiny $\{2, 3, \ldots, n^2\}$, která obsahuje

$$\pi(n^2) \approx \frac{n^2}{\ln n^2}$$

prvočísel. Musíme zjistit, kolik z těchto prvočísel je dělitelem $w$.

# Randomizovaný komunikační protokol

Zjevně je $w = |Num(x) - Num(y)| < 2^n$. Číslo $w$ můžeme rozložit na prvočísla

$$w = p_1^{i_1} p_2^{i_2} \cdots p_k^{i_k}$$

Chceme ukázat, že $k \leq n - 1$.

Předpokládejme, že $k \geq n$. Pak

$$w = p_1^{i_1} p_2^{i_2} \cdots p_k^{i_k} \geq p_1 p_2 \cdots p_n > 1 \cdot 2 \cdot 3 \cdot \cdots \cdot n = n! > 2^n$$

což je spor s tím, že $w < 2^n$.

Pravděpodobnost, že z množiny $\{2, 3, \cdots, n^2\}$ vybereme náhodně prvočíslo, které je dělitelem $w$ je maximálně

$$\frac{n-1}{\pi(n^2)} \leq \frac{n-1}{n^2 / \ln n^2} \leq \frac{\ln n^2}{n}$$

Například pro $n = 10^{16}$ je to maximálně $0.36892 \cdot 10^{-14}$.

# Randomizovaný komunikační protokol

Modifikace protokolu:

- $C_1$ vygeneruje náhodně 10 prvočísel $p_1, p_2, \ldots, p_{10}$ a pošle

$$p_1, s_1, p_2, s_2, \ldots, p_{10}, s_{10}$$

  kde $s_i = Num(x) \bmod p_i$.

- $C_2$ spočítá $t_i = Num(y) \bmod p_i$ pro $i \in \{1, \ldots, 10\}$.

  Pokud $s_i \neq t_i$ pro nějaké $i$, vydá odpověď „$x \neq y$“, jinak vydá odpověď „$x = y$“.

Pro $n = 10^{16}$ je třeba přenést maximálně $2560$ bitů.

# Randomizovaný komunikační protokol

Protože 10 prvočísel je vybíráno náhodně, je pravděpodobnost chyby maximálně

$$\left(\frac{n-1}{\pi(n^2)}\right)^{10} \leq \left(\frac{\ln n^2}{n}\right)^{10} = \frac{2^{10} \cdot (\ln n)^{10}}{n^{10}}$$

Pro $n = 10^{16}$ je pravděpodobnost chyby menší než $0.4717 \cdot 10^{-141}$.

# Aproximační algoritmy

# Aproximační algoritmy

Mnoho důležitých problémů je NP-úplných. U optimalizačních problémů často nepotřebujeme nalézt nejoptimálnější řešení, ale stačí nám řešení, které je „dostatečně dobré“.

Algoritmům, které vrací takováto „dostatečně dobrá“ řešení, se říká **aproximační algoritmy**.

**Poznámka:** **Optimalizační problém** je problém, kde pro každý vstup $w$ máme definovánu množinu $S_w$ všech **přípustných řešení** a funkci $c : S_w \to \mathbb{R}$.

Úkolem je najít takové $x \in S_w$, pro které je hodnota $c(x)$ minimální (resp. maximální).

**Příklady:** Hledání nejkratší cesty v grafu, určování maximálního toku v síti.

# Aproximační algoritmy

Řekněme, že řešíme problém, kde je úkolem hodnotu $c(x)$ minimalizovat.

Řekněme, že pro vstup $w$ vydá algoritmus řešení $x$, přičemž optimálním řešením je $x^*$. Zjevně platí $c(x) \geq c(x^*)$.

Jako **aproximační poměr** daného algoritmu označujeme funkci $\rho(n)$, která každému $n$ přiřazuje maximální hodnotu poměru $c(x)/c(x^*)$ pro všechny vstupy velikosti $n$.

**Poznámka:** Naopak u algoritmu, kde je cílem hodnotu $c(x)$ maximalizovat, bychom uvažovali poměr $c(x^*)/c(x)$.

# Aproximační algoritmy

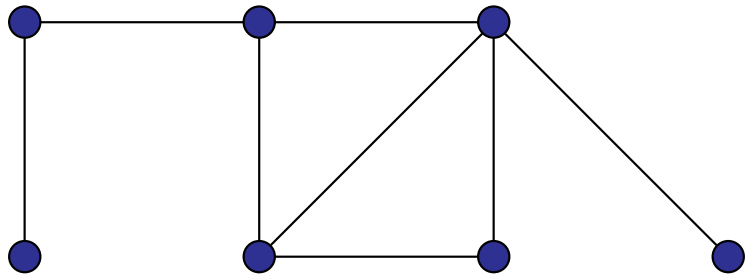
O algoritmu, který dosahuje aproximačního poměru $\rho(n)$, říkáme, že je to $\rho(n)$-**aproximační algoritmus**.

Pro řadu problémů jsou známy polynomiální algoritmy, kde je hodnota $\rho(n)$ omezena:

- malou konstantou, např. 2-aproximační algoritmy, nebo
- pomalu rostoucí funkcí, např. $\Theta(\log n)$-aproximační algoritmy.

# Vrcholové pokrytí grafu

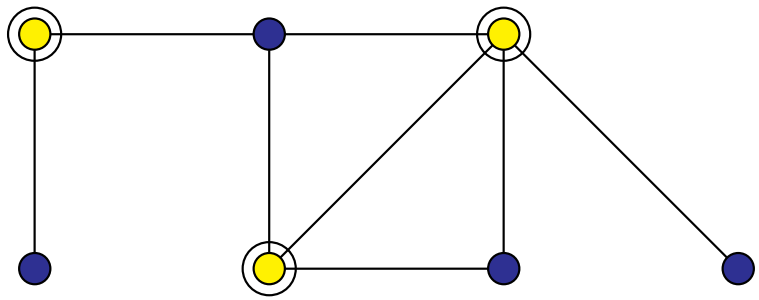
Mějme neorientovaný graf $G = (V, E)$. **Vrcholové pokrytí** (vertex-cover) grafu $G$ je množina $C \subseteq V$ taková, že pro každou hranu $(u, v) \in E$ platí, že buď $u \in C$ nebo $v \in C$.



Úkolem je najít pro zadaný graf minimální vrcholové pokrytí.

# Vrcholové pokrytí grafu

Mějme neorientovaný graf $G = (V, E)$. **Vrcholové pokrytí** (vertex-cover) grafu $G$ je množina $C \subseteq V$ taková, že pro každou hranu $(u, v) \in E$ platí, že buď $u \in C$ nebo $v \in C$.



Úkolem je najít pro zadaný graf minimální vrcholové pokrytí.

# Vrcholové pokrytí grafu

Máme tedy vyřešit následující problém:

## Minimální vrcholové pokrytí grafu (vertex cover)

Vstup: Neorientovaný graf $G = (V, E)$.

Výstup: Minimalní množina $C$ ($C \subseteq V$) tvořící vrcholové pokrytí grafu $G$.

# Vrcholové pokrytí grafu

Máme tedy vyřešit následující problém:

## Minimální vrcholové pokrytí grafu (vertex cover)

Vstup: Neorientovaný graf $G = (V, E)$.

Výstup: Minimalní množina $C$ ($C \subseteq V$) tvořící vrcholové pokrytí grafu $G$.

Následující (rozhodovací) varianta tohoto problému je NP-úplná:

## Vrcholové pokrytí (vertex cover)

Vstup: Neorientovaný graf $G = (V, E)$ a přirozené číslo $k$.

Otázka: Existuje vrcholové pokrytí grafu $G$ tvořené $k$ vrcholy?

Tento problém tedy není možné řešit v polynomiálním čase (leda by platilo P = NP).

# Vrcholové pokrytí grafu

Máme tedy vyřešit následující problém:

## Minimální vrcholové pokrytí grafu (vertex cover)

Vstup: Neorientovaný graf $G = (V, E)$.

Výstup: Minimalní množina $C$ ($C \subseteq V$) tvořící vrcholové pokrytí grafu $G$.

Existuje však 2-aproximační algoritmus řešící tento problém:

- Algoritmus najde pro zadaný graf $G$ vrcholové pokrytí $C$ takové, že

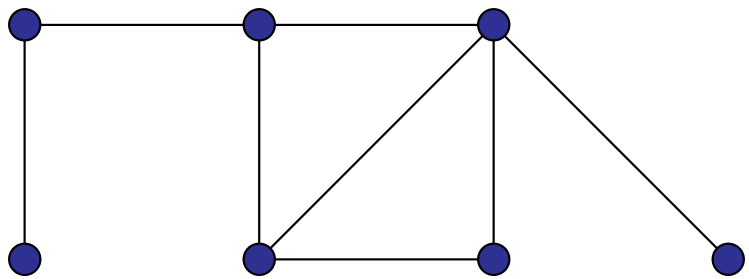
$$|C| \leq 2k$$

kde $k$ je velikost minimálního vrcholového pokrytí grafu $G$.

# Vrcholové pokrytí grafu

APPROX-VERTEX-COVER ($G$):

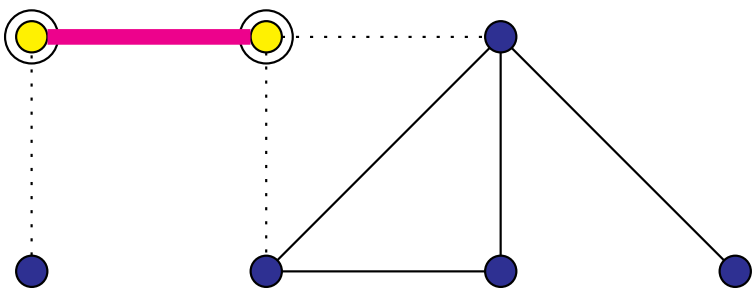
- $C := \emptyset$
- $E' := E$
- **while** $E' \neq \emptyset$ **do**
  - vyber libovolnou hranu $(u, v)$ z $E'$
  - $C := C \cup \{u, v\}$
  - odstraň z $E'$ hrany incidentní s $u$ nebo $v$
- **return** $C$



# Vrcholové pokrytí grafu

APPROX-VERTEX-COVER ($G$):

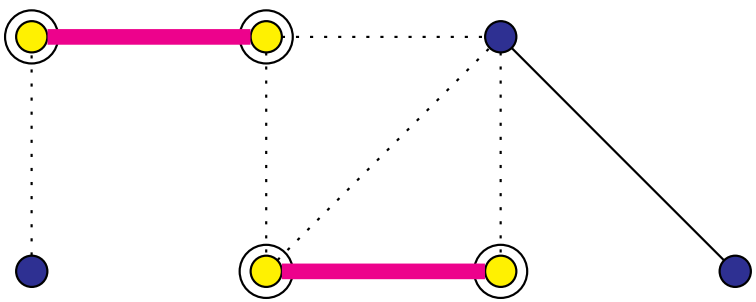
- $C := \emptyset$
- $E' := E$
- **while** $E' \neq \emptyset$ **do**
  - vyber libovolnou hranu $(u, v)$ z $E'$
  - $C := C \cup \{u, v\}$
  - odstraň z $E'$ hrany incidentní s $u$ nebo $v$
- **return** $C$



# Vrcholové pokrytí grafu

APPROX-VERTEX-COVER ($G$):

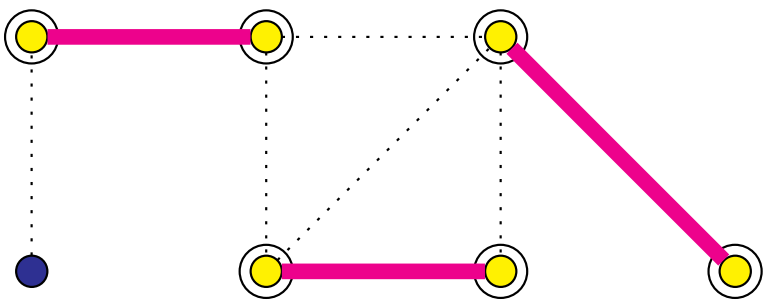
- $C := \emptyset$
- $E' := E$
- **while** $E' \neq \emptyset$ **do**
  - vyber libovolnou hranu $(u, v)$ z $E'$
  - $C := C \cup \{u, v\}$
  - odstraň z $E'$ hrany incidentní s $u$ nebo $v$
- **return** $C$



# Vrcholové pokrytí grafu

APPROX-VERTEX-COVER ($G$):

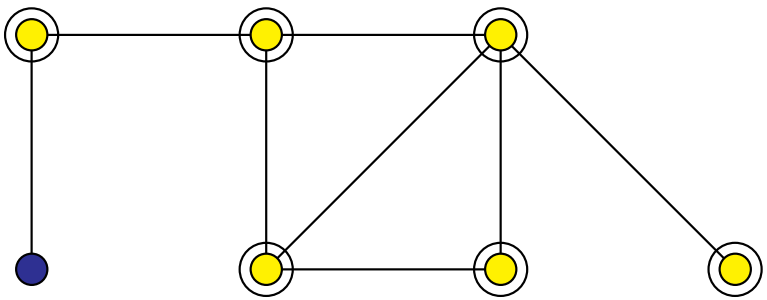
- $C := \emptyset$
- $E' := E$
- **while** $E' \neq \emptyset$ **do**
  - vyber libovolnou hranu $(u, v)$ z $E'$
  - $C := C \cup \{u, v\}$
  - odstraň z $E'$ hrany incidentní s $u$ nebo $v$
- **return** $C$



# Vrcholové pokrytí grafu

APPROX-VERTEX-COVER ($G$):

- $C := \emptyset$
- $E' := E$
- **while** $E' \neq \emptyset$ **do**
  - vyber libovolnou hranu $(u, v)$ z $E'$
  - $C := C \cup \{u, v\}$
  - odstraň z $E'$ hrany incidentní s $u$ nebo $v$
- **return** $C$



# Vrcholové pokrytí grafu

**Tvrzení**

APPROX-VERTEX-COVER je polynomiální 2-aproximační algoritmus.

**Důkaz:** Označme $A$ množinu všech hran vybraných v kroku 4, $C$ vrcholové pokrytí nalezené algoritmem a $C^*$ minimální vrcholové pokrytí grafu $G$.

Jakékoliv vrcholové pokrytí musí obsahovat alespoň jeden z koncových vrcholů každé hrany z $A$.

Pro libovolné vrcholové pokrytí $C'$ tedy platí $|A| \leq |C'|$.

Speciálně pro $C^*$ tedy také musí platit $|A| \leq |C^*|$.

Na druhou stranu, zjevně platí $|C| = 2 \cdot |A|$.
Dohromady tedy dostáváme $|C| \leq 2 \cdot |C^*|$.

# Problém obchodního cestujícího (TSP)

## Problém obchodního cestujícího (TSP)

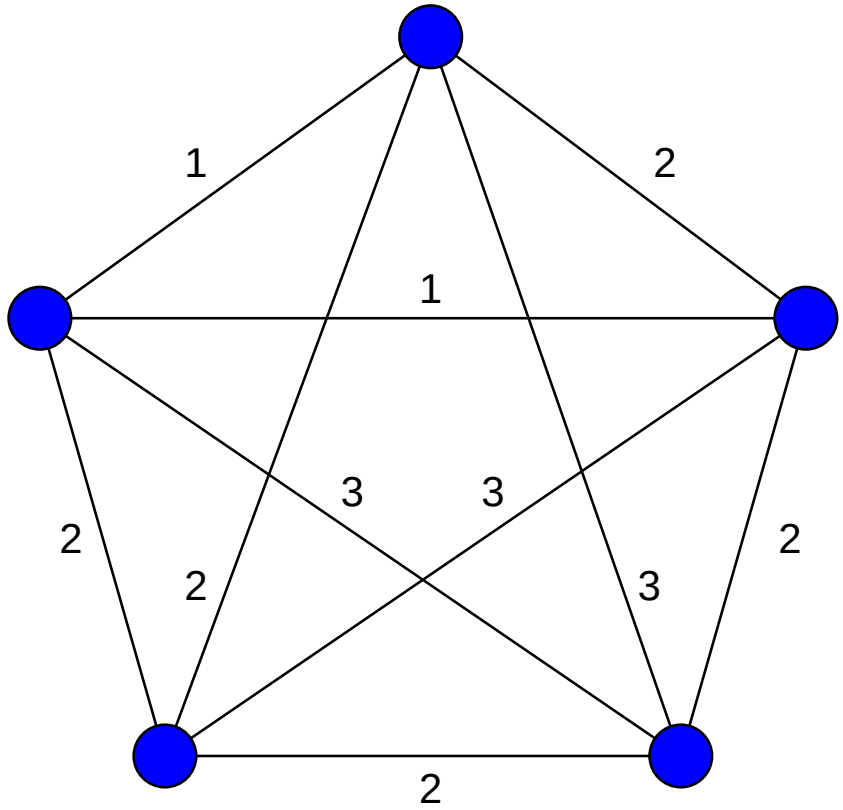
Vstup: Množina $n$ měst a vzdálenosti mezi nimi.

Výstup: Nejkratší okružní cesta procházející všemi městy.

**Poznámka:** Slovem „okružní“ myslíme, že cesta končí ve stejném městě, kde začíná.

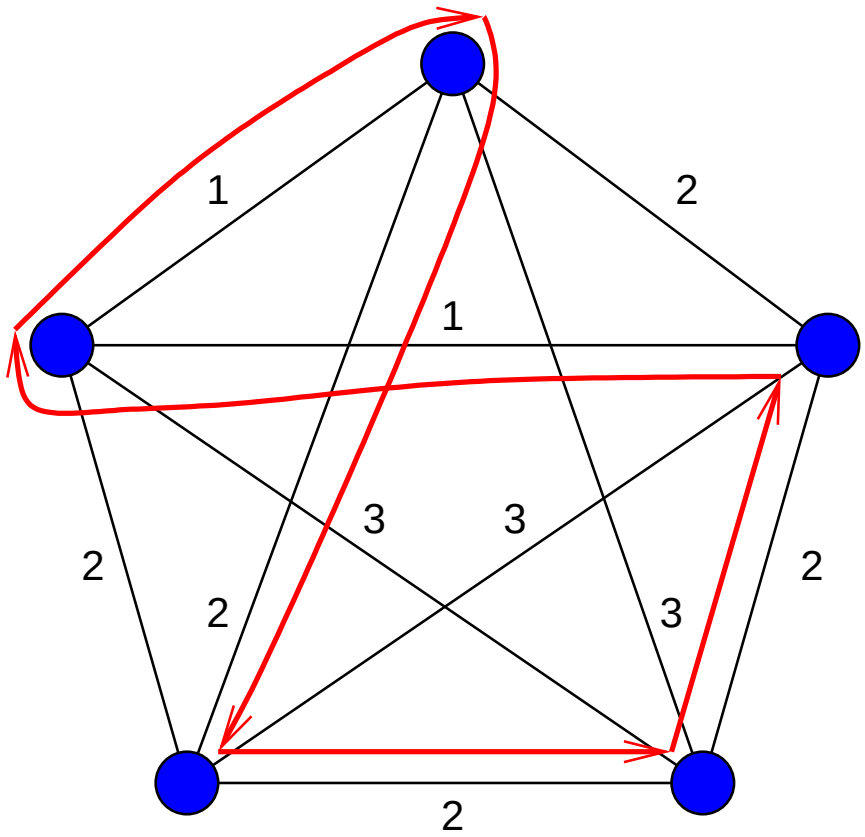
# Problém obchodního cestujícího (TSP)

Příklad instance problému:



# Problém obchodního cestujícího (TSP)

Příklad instance problému:




Nejkratší okružní cesta má délku 8.

# Problém obchodního cestujícího (TSP)

Můžeme uvažovat dvě různé varianty tohoto problému:

- Každé město musí být navštíveno právě jednou.
- Města je možné navštěvovat opakovaně.

# Problém obchodního cestujícího (TSP)

V následujícím výkladu se nám bude hodit poněkud formálnější definice problému:

Na instanci problému (tj. množinu měst a vzdálenosti mezi nimi) se můžeme dívat jako na úplný neorientovaný graf $G = (V, E)$ s ohodnocením hran $d$ (kde $d : E \to \mathbb{N}$).

Pro libovolnou množinu hran $E' \subseteq E$ definujeme

$$d(E') = \sum_{e \in E'} d(e)$$

Pro libovolný cyklus $H$ pak definujeme $d(H) = d(E')$, kde $E'$ je množina hran ležících na cyklu $H$.

# Problém obchodního cestujícího (TSP)

Problém TSP pak můžeme formulovat následovně:

## Problém obchodního cestujícího (TSP)

Vstup: Úplný neorientovaný graf $G = (V, E)$ s ohodnocením hran $d$.

Výstup: Cyklus $H$ procházející všemi vrcholy grafu $G$ takový, že hodnota $d(H)$ je minimální možná.

# Problém obchodního cestujícího (TSP)

Následující problém je NP-úplný (ať už je či není povoleno navštěvovat vrcholy opakovaně):

## Problém obchodního cestujícího (TSP) – rozhodovací varianta

Vstup: Úplný neorientovaný graf $G = (V, E)$ s ohodnocením hran $d$ a číslo $L$.

Otázka: Existuje v grafu $G$ cyklus $H$ procházející všemi vrcholy takový, že $d(H) \leq L$?

**Poznámka:** Nemůžeme tedy očekávat, že by problém TSP (ať už v té či oné variantě) byl řešitelný v polynomiálním čase, leda že by platilo P = NP.

# Problém obchodního cestujícího

Pro problém TSP, kde vyžadujeme aby byl každý vrchol navštíven právě jednou, se dá dokázat následující:

**Věta**

Pokud $P \neq NP$, pak pro žádnou konstantu $\rho \geq 1$ neexistuje polynomiální $\rho$-aproximační algoritmus řešící problém obchodního cestujícího.

**Důkaz:** Předpokládejme, že by pro nějaké $\rho$ takový algoritmus existoval. Ukážeme, že pak by existoval polynomiální algoritmus pro problém Hamiltonovské kružnice.

Problém Hamiltonovské kružnice je NP-úplný, nalezení polynomiálního algoritmu by znamenalo, že $P = NP$.

# Problém obchodního cestujícího

Nechť $G = (V, E)$ je instance problému Hamiltonovské kružnice.

Instanci problému obchodního cestujícího $G' = (V', E')$ sestrojíme tak, že $V' = V$ a $E'$ obsahuje všechny možné hrany, kterým přiřadíme ohodnocení

$$d(u, v) = \begin{cases} 1 & \text{pokud } (u, v) \in E \\ \rho \cdot |V| + 1 & \text{jinak} \end{cases}$$

Jestliže graf $G$ obsahuje Hamiltonovskou kružnici, pak v $G'$ existuje okružní cesta délky $|V|$.

Jakákoliv okružní cesta v $G'$, která obsahuje hranu, která není v $G$, má délku větší než $\rho \cdot |V|$.

Pokud v $G$ Hamiltonovská kružnice existuje, $\rho$-aproximační algoritmus musí nějakou takovou kružnici vrátit jako výslednou okružní cestu.

# Problém obchodního cestujícího (TSP)

Není těžké si rozmyslet, že varianta TSP kde je povoleno opakovaně navštěvovat vrcholy se dá snadno převést na variantu, kde musí být každý vrchol navštíven právě jednou:

- Pro daný graf $G$ s ohodnocením $d$ sestrojíme nové ohodnocení $d'$, kde $d'(u, v)$ je délka nejkratší cesty z $u$ do $v$

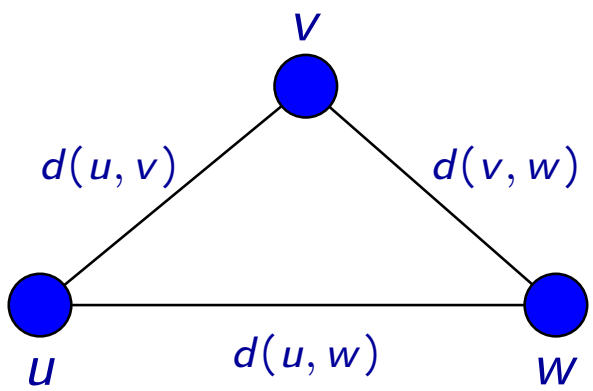
**Poznámka:** Pro nalezení nejkratších cest mezi dvojicemi vrcholů existují rychlé polynomiální algoritmy (např. Dijkstrův, Floydův-Warshallův apod.)

Graf s ohodnocením $d'$ navíc splňuje tzv. **trojúhelníkovou nerovnost**.

# Problém obchodního cestujícího (TSP)

V grafu $G$ s ohodnocením $d$ je splněna **trojúhelníková nerovnost**, jestliže pro libovolnou trojici jeho vrcholů $u, v, w$ platí

$$d(u, w) \leq d(u, v) + d(v, w)$$




Tj. nejkratší cesta z $u$ do $w$ je vždy po hraně $(u, w)$ a nemá cenu jít „oklikou“ přes nějaký jiný vrchol.

# Problém obchodního cestujícího (TSP)

Variantu TSP, ve které se omezujeme pouze na instance, ve kterých je splněna trojúhelníková nerovnost (a kde musí být každý vrchol navštíven právě jednou), označujeme **$\Delta$-TSP**.

# Problém obchodního cestujícího (TSP)

Variantu TSP, ve které se omezujeme pouze na instance, ve kterých je splněna trojúhelníková nerovnost (a kde musí být každý vrchol navštíven právě jednou), označujeme **$\Delta$-TSP**.

Pro problém $\Delta$-TSP je znám 1.5-aproximační polynomiální algoritmus, tj. algoritmus, který pro daný graf $G$ s ohodnocením $d$ vrátí cyklus $H$ procházející všemi vrcholy takový, že

$$d(H) \leq 1.5 \cdot d(H^*)$$

kde $H^*$ optimální řešení (tj. cyklus s minimální hodnotou $d(H^*)$).

# Problém obchodního cestujícího (TSP)

Variantu TSP, ve které se omezujeme pouze na instance, ve kterých je splněna trojúhelníková nerovnost (a kde musí být každý vrchol navštíven právě jednou), označujeme $\Delta$-**TSP**.

Pro problém $\Delta$-TSP je znám 1.5-aproximační polynomiální algoritmus, tj. algoritmus, který pro daný graf $G$ s ohodnocením $d$ vrátí cyklus $H$ procházející všemi vrcholy takový, že

$$d(H) \leq 1.5 \cdot d(H^*)$$

kde $H^*$ optimální řešení (tj. cyklus s minimální hodnotou $d(H^*)$).

My si ukážeme poněkud jednodušší 2-aproximační polynomiální algoritmus pro problém $\Delta$-TSP.

# Problém obchodního cestujícího (TSP)

Před vlastním popisem algoritmu si připomeňme některé pojmy:

**Kostra** grafu $G = (V, E)$ je libovolný souvislý acyklický graf $T = (V', E')$, kde $V = V'$ a $E' \subseteq E$ (tj. $T$ je souvislý podgraf grafu $G$ obsahující všechny vrcholy z $G$).

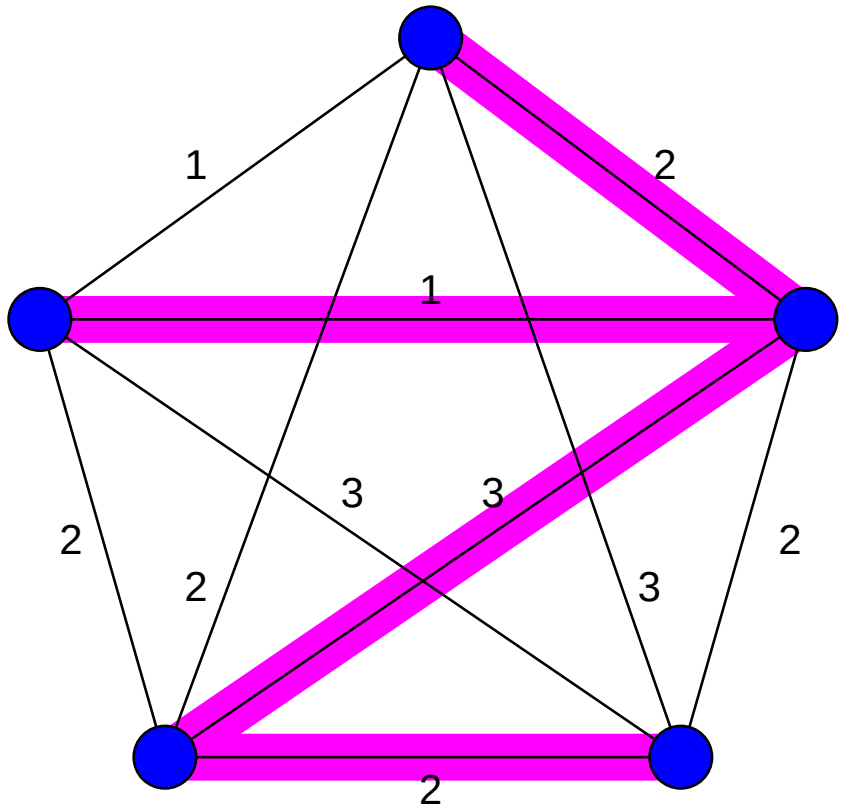
Hodnotu $d(T)$ definujeme jako součet hodnot hran v této kostře, tj. $d(T) = d(E')$.

Kostra $T$ je **minimální**, jestliže pro libovolnou jinou kostru $T'$ v grafu $G$ platí $d(T) \leq d(T')$.

Pro problém nalezení minimální kostry v daném ohodnoceném grafu jsou známy rychlé polynomiální algoritmy (např. Kruskalův nebo Jarníkův (Primův)).

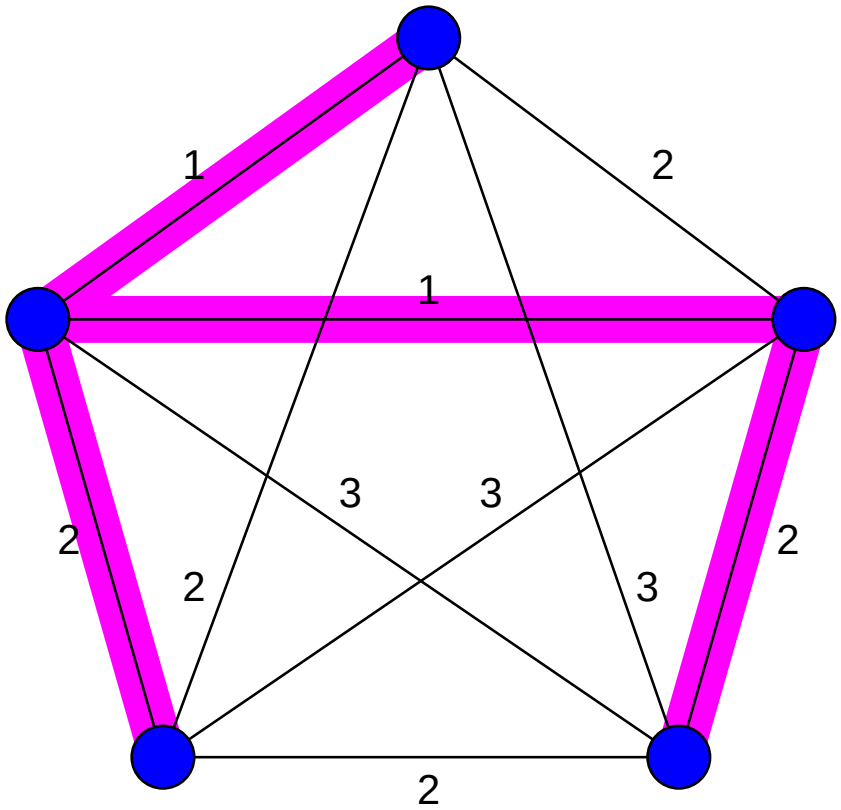
# Problém obchodního cestujícího (TSP)

Příklad kostry $T$, kde $d(T) = 8$:



# Problém obchodního cestujícího (TSP)

Příklad minimální kostry $T$, kde $d(T) = 6$:



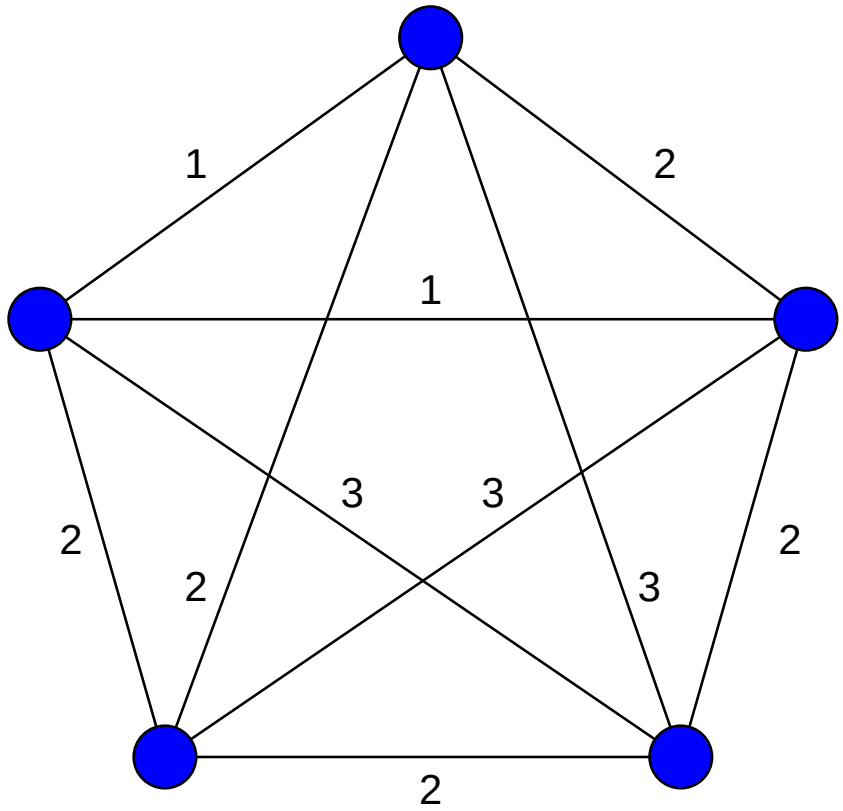
# Problém obchodního cestujícího (TSP)

2-aproximační algoritmus pro $\Delta$-TSP pracuje v následujících krocích:

- Najde minimální kostru grafu $G$.
- Vytvoří uzavřený tah podél této kostry.
- Z vytvořeného tahu odstraní opakující se vrcholy a výsledný cyklus vrátí jako výsledek.

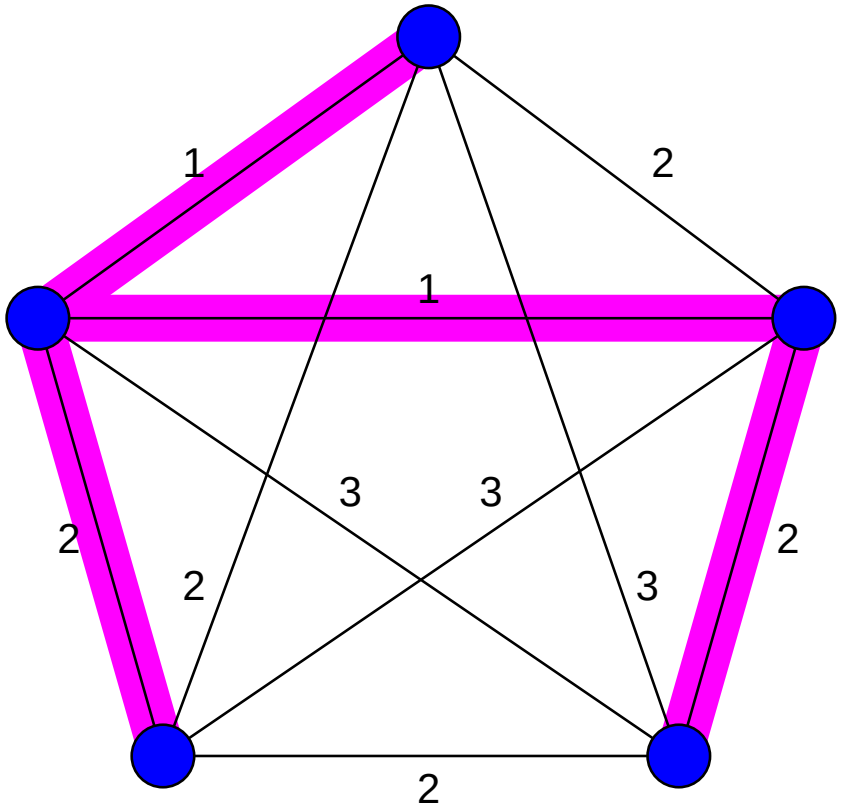
# Problém obchodního cestujícího (TSP)

Vezměme si následující instanci $\Delta$-TSP.



# Problém obchodního cestujícího (TSP)

Krok 1: Nalezení minimální kostry $T$



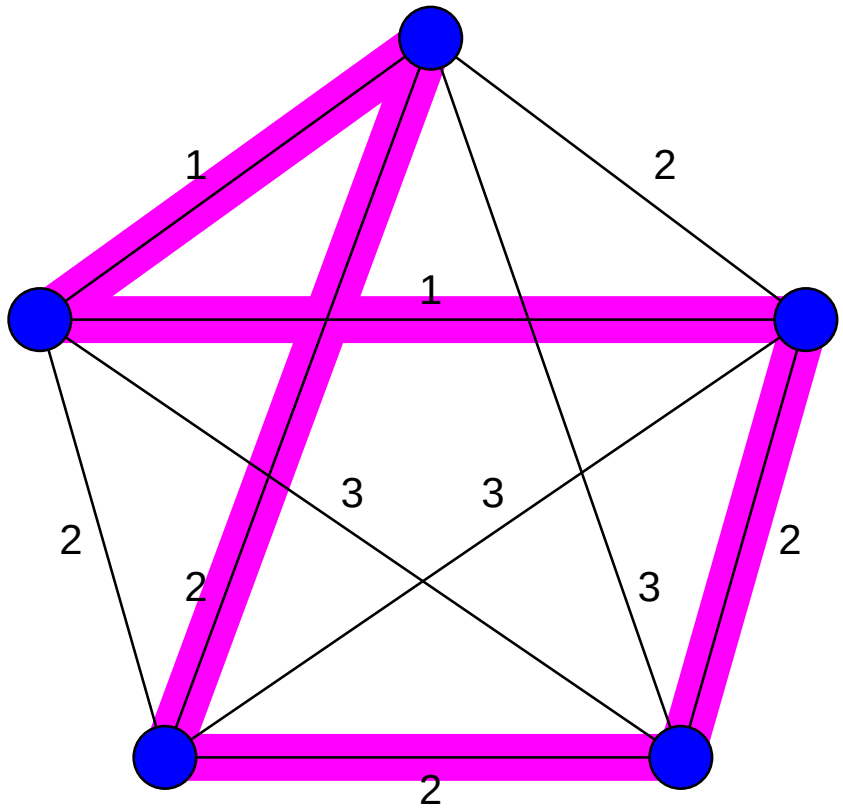
# Problém obchodního cestujícího (TSP)

Všimněme si, že $d(T) < d(H^*)$:

- Pokud z $H^*$ odstraníme libovolnou hranu, dostaneme kostru $T'$. Zjevně platí $d(T') < d(H^*)$.
- Pro libovolnou kostru $T'$ platí $d(T) \leq d(T')$.

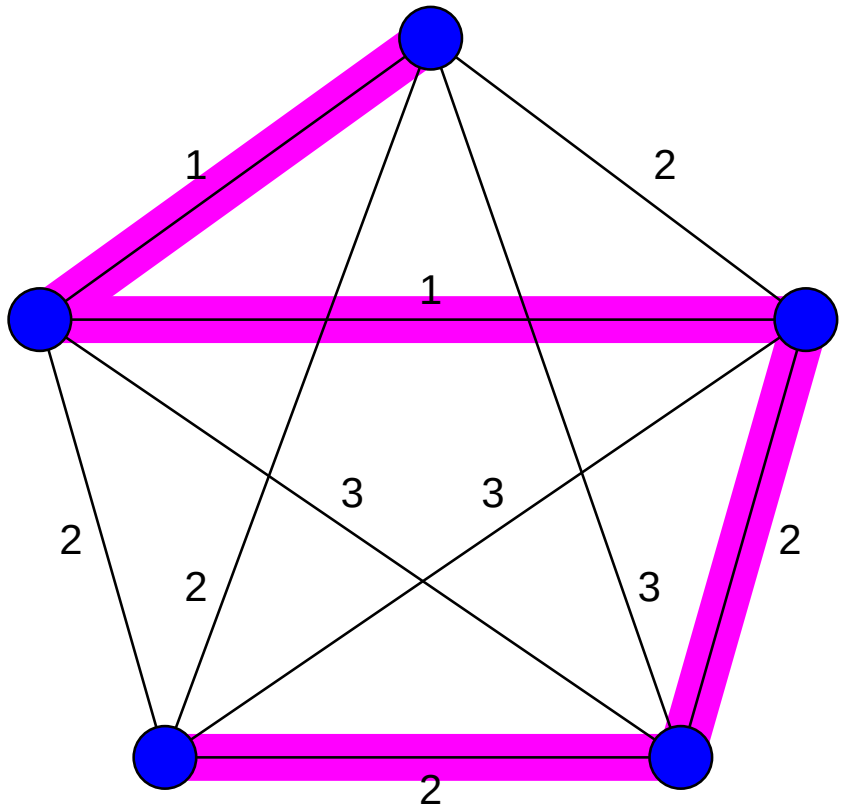
# Problém obchodního cestujícího (TSP)

Příklad: Vezměme si optimální cyklus



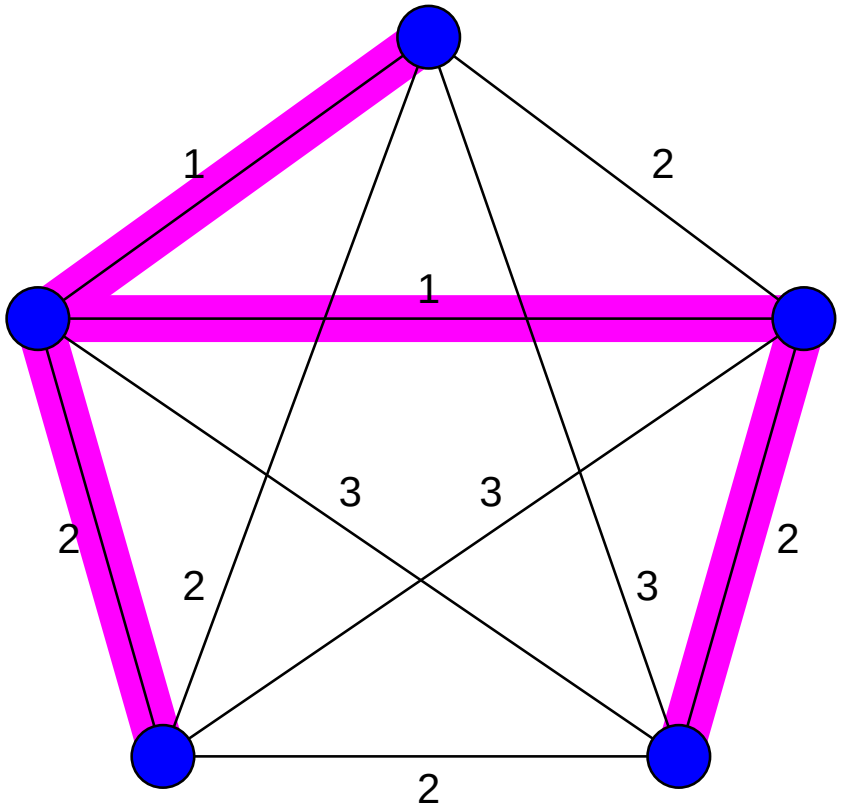
# Problém obchodního cestujícího (TSP)

Příklad: Odstraněním jedné hrany vznikne kostra



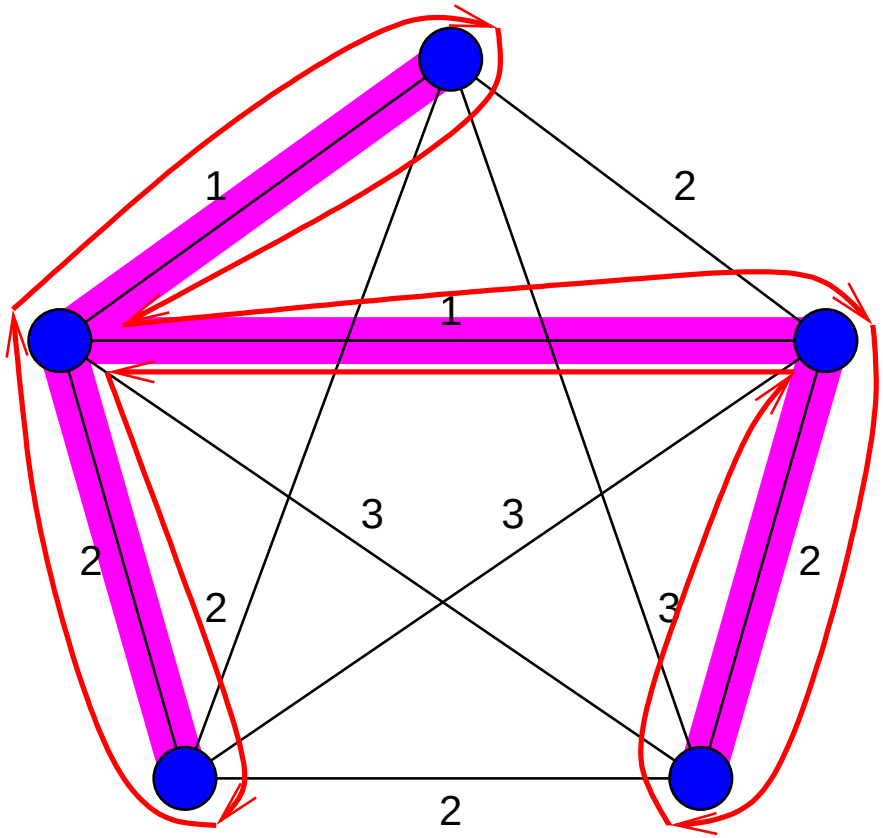
# Problém obchodního cestujícího (TSP)

Krok 1: Nalezení minimální kostry



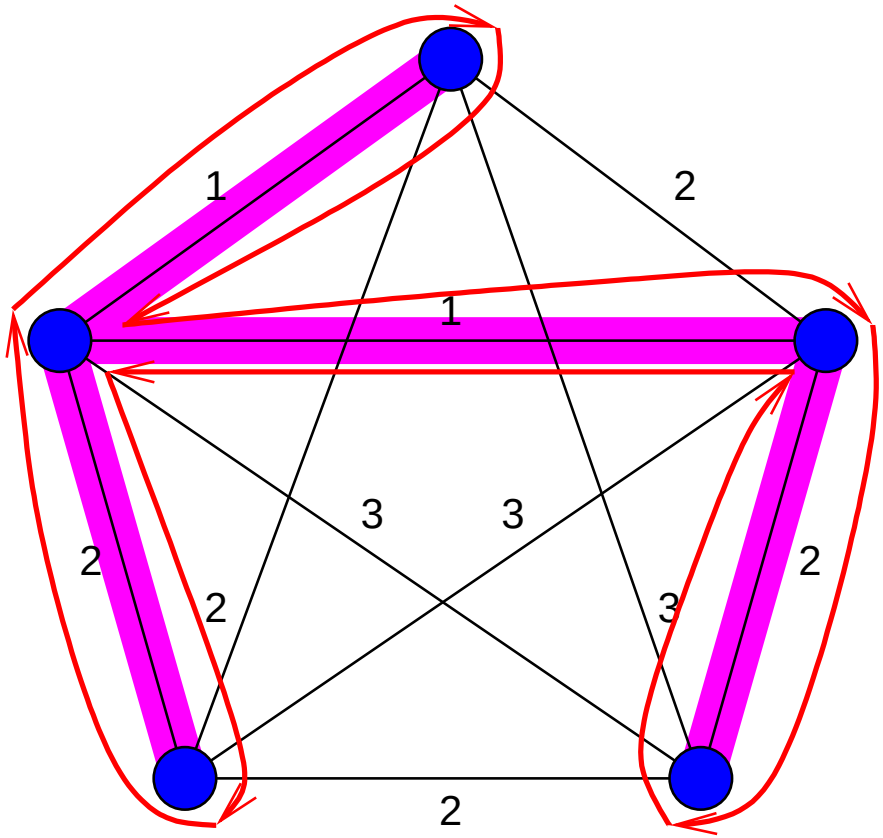
# Problém obchodního cestujícího (TSP)

Krok 2: Vytvoření tahu $H$ podél kostry



# Problém obchodního cestujícího (TSP)

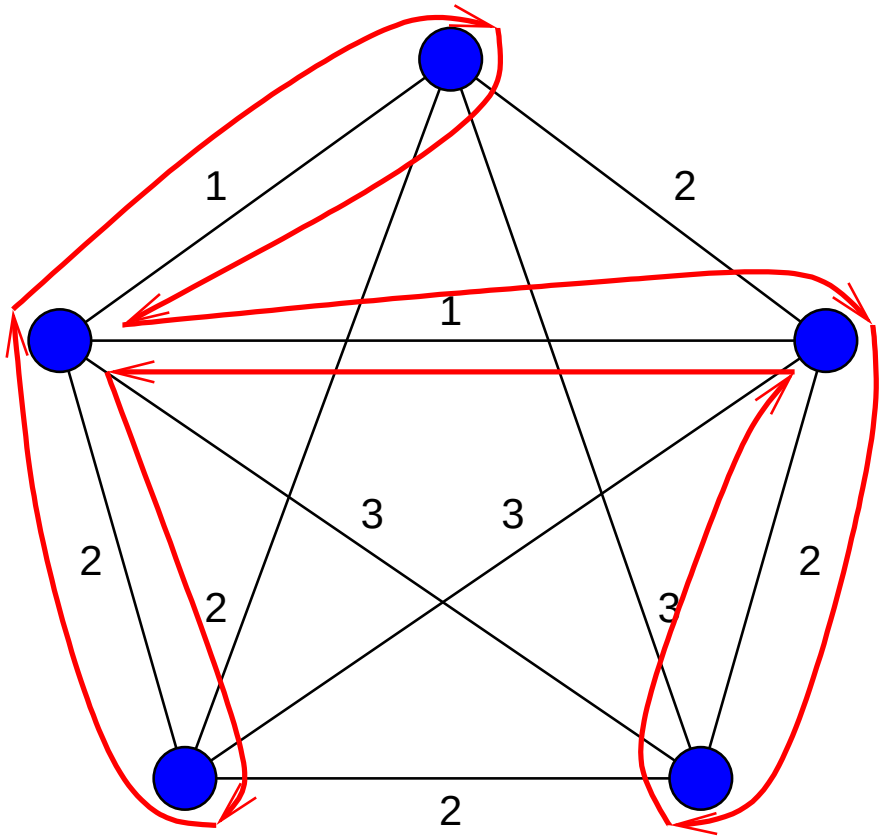
Krok 2: Vytvoření tahu $H$ podél kostry




Každou hranu procházíme dvakrát, platí tedy $d(H) = 2d(T) < 2d(H^*)$.

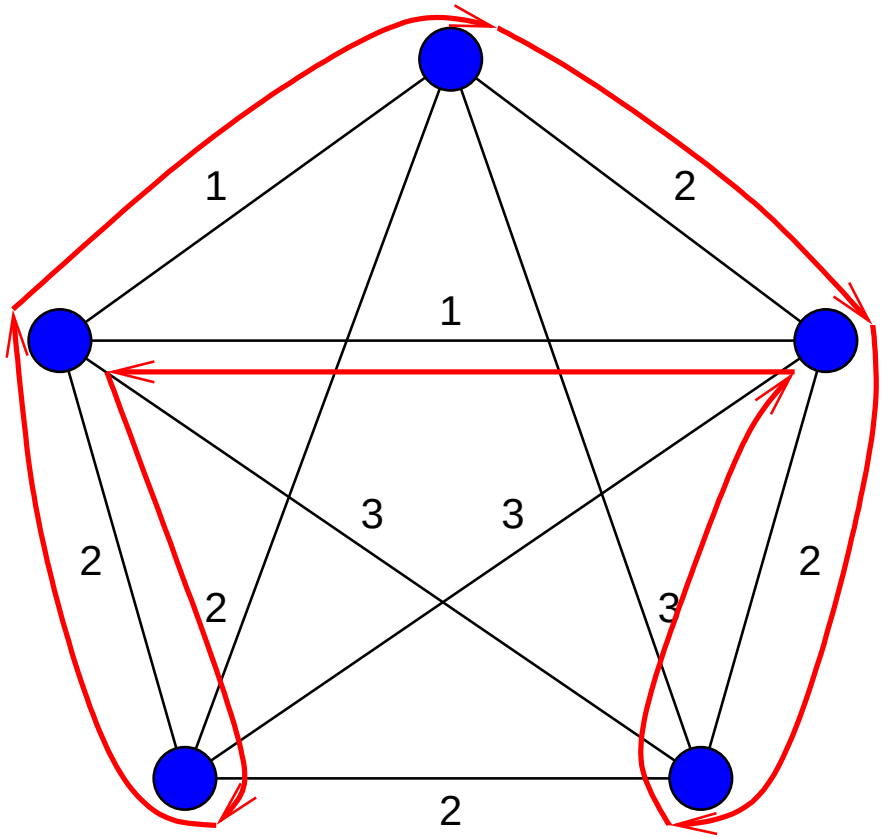
# Problém obchodního cestujícího (TSP)

Krok 2: Vytvoření tahu $H$ podél kostry



# Problém obchodního cestujícího (TSP)

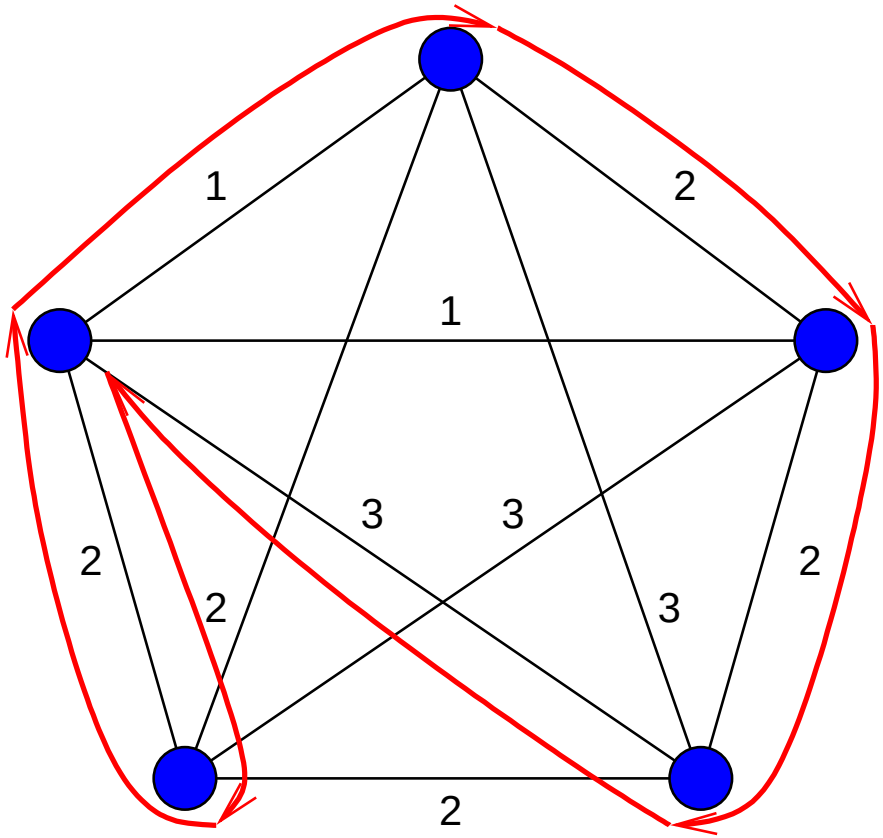
Krok 3: Postupné vypouštění opakujících se vrcholů z tahu $H$




Každé vypuštění vrcholu tah leda zkrátí.

# Problém obchodního cestujícího (TSP)

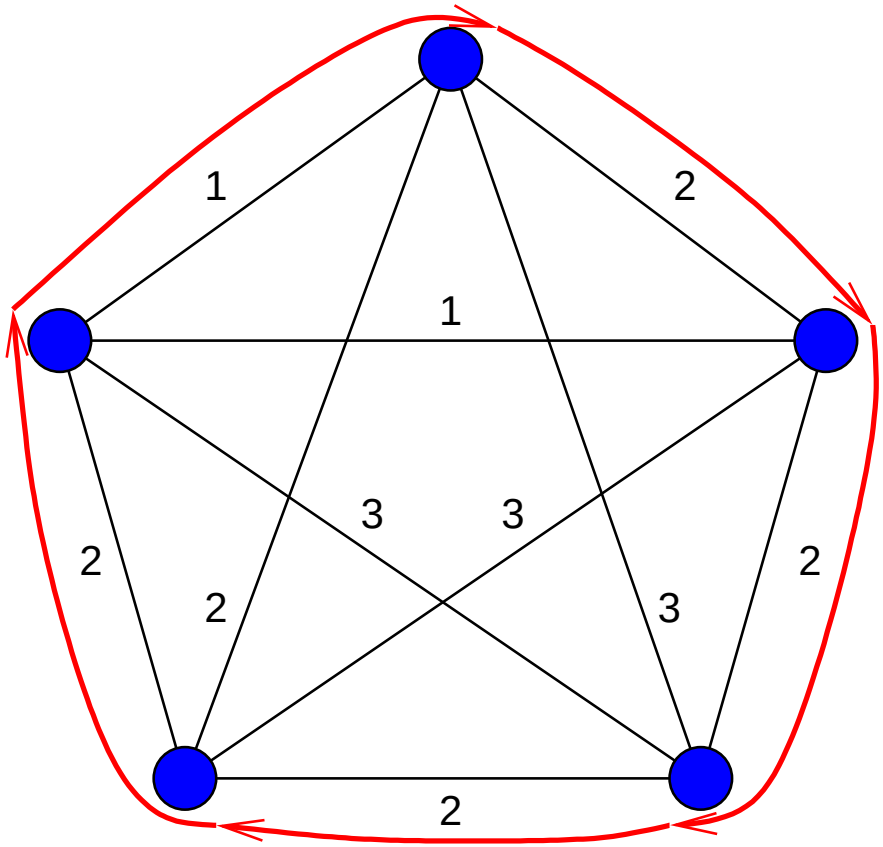
Krok 3: Postupné vypouštění opakujících se vrcholů z tahu $H$




Každé vypuštění vrcholu tah leda zkrátí.

# Problém obchodního cestujícího (TSP)

Krok 3: Postupné vypouštění opakujících se vrcholů z tahu $H$




Každé vypuštění vrcholu tah leda zkrátí.

# Problém obchodního cestujícího (TSP)

Nalezený cyklus: