Řešení těžkých 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 těžkých 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 těžkých 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.

Řešení těžkých problémů

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. $m = 10000$).

Řešení těžkých problémů

Problém „SAT“

Vstup: Booleovská formule ϕ v KNF.

Otázka: Je ϕ splnitelná?

Pokud ϕ obsahuje pouze Hornovy klauzule, je možné SAT vyřešit 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

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

Řešení těžkých problémů

SAT je možné vyřešit v polynomiálním čase také v případě, kdy se omezíme na formule, kde každá klauzule obsahuje nanejvýš dva literály.

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 těžkých 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 ε.

- **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).

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.

V roce 2002 se podařilo najít polynomiální algoritmus se složitostí $O(n^{12})$ (N. Saxona, M. Agrawal, N. Kayal).

Později se ho podařilo zlepšit na $O(n^8)$, ale pro praktické účely je stále nepoužitelný.

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

- Miller-Rabin (1976)
- Solovay-Strassen (1977)

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* ε.

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.

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 $\{1, 2, \ldots, n-1\}$:

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

```
1  d ← 1
2  ⟨b_k, b_{k−1}, ..., b_0⟩ je binární reprezentace b
3  for i ← k downto 0
4        do d ← (d · d) mod n
5           if b_i = 1
6              then d ← (d · a) mod n
7  return d
```

Testování prvočíselnosti (Miller-Rabin)

Na umocňování je založen i následující algoritmus pro testování prvočíselnosti:

MILLER-RABIN(n,s)

```
1  for j ← 1 to s
2        do a ← RANDOM(2, n − 1)
3           if WITNESS(a, n)
4              then return „Složené“        ▷ n je zcela jistě složené.
5  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)

```
 1  d ← 1
 2  ⟨b_k, b_{k−1}, ..., b_0⟩ je binární reprezentace n − 1
 3  for i ← k downto 0
 4        do x ← d
 5           d ← (d · d) mod n
 6           if d = 1 and x ≠ 1 and x ≠ n − 1
 7              then return TRUE
 8           if b_i = 1
 9              then d ← (d · a) mod n
10  if d ≠ 1
11     then return TRUE
12  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.

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 „$x = y$“.

Randomizovaný komunikační protokol

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

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

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

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

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 cíle hodnotu $c(x)$ maximalizovat, bychom uvažovali poměr $c(x^*)/c(x)$.

Aproximační algoritmy

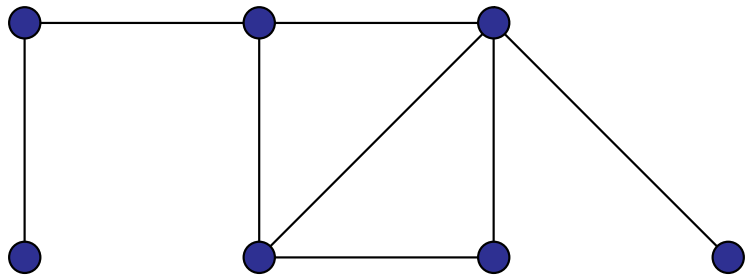
O algoritmu, který dosahuje aproximačního poměru $\rho(n)$, říkáme, že je to $\rho(n)$-**aproximační algoritmus**.

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 $v \in C$.

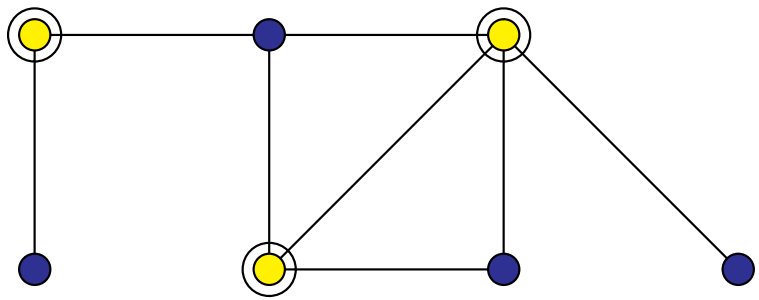


Úkolem je najít pro zadaný graf minimální vrcholové pokrytí.

Poznámka: Je známo, že se jedná o NP-úplný problém.

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 $v \in C$.



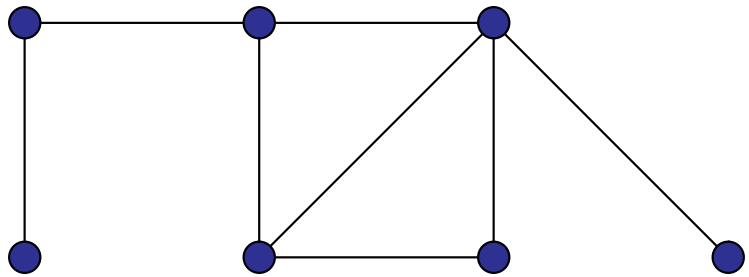
Úkolem je najít pro zadaný graf minimální vrcholové pokrytí.

Poznámka: Je známo, že se jedná o NP-úplný problém.

Vrcholové pokrytí grafu

APPROX-VERTEX-COVER(G)

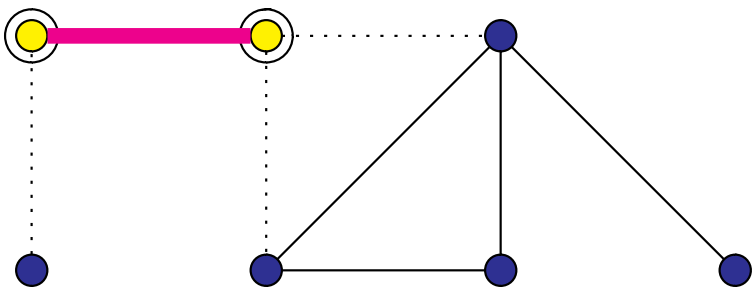
1 $C \leftarrow \emptyset$
2 $E' \leftarrow E$
3 **while** $E' \neq \emptyset$
4 **do** vyber libovolnou hranu (u, v) z E'
5 $C \leftarrow C \cup \{u, v\}$
6 odstraň z E' hrany incidentní s u nebo v
7 **return** C



Vrcholové pokrytí grafu

APPROX-VERTEX-COVER(G)

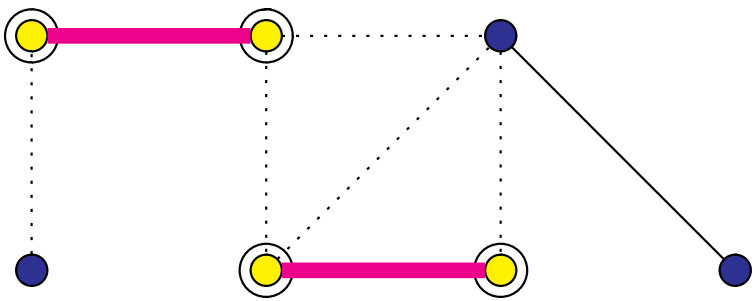
```
1  C ← ∅
2  E' ← E
3  while E' ≠ ∅
4        do vyber libovolnou hranu (u, v) z E'
5           C ← C ∪ {u, v}
6           odstraň z E' hrany incidentní s u nebo v
7  return C
```



Vrcholové pokrytí grafu

APPROX-VERTEX-COVER(G)

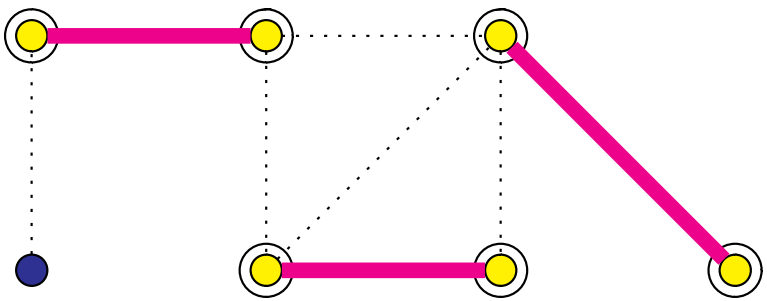
1 $C \leftarrow \emptyset$
2 $E' \leftarrow E$
3 **while** $E' \neq \emptyset$
4 **do** vyber libovolnou hranu (u, v) z E'
5 $C \leftarrow C \cup \{u, v\}$
6 odstraň z E' hrany incidentní s u nebo v
7 **return** C



Vrcholové pokrytí grafu

APPROX-VERTEX-COVER(G)

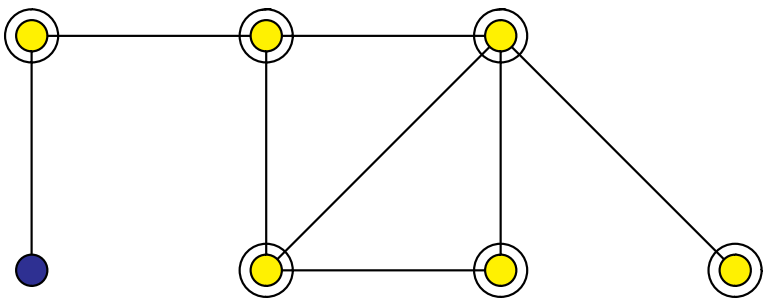
1 $C \leftarrow \emptyset$
2 $E' \leftarrow E$
3 **while** $E' \neq \emptyset$
4 **do** vyber libovolnou hranu (u, v) z E'
5 $C \leftarrow C \cup \{u, v\}$
6 odstraň z E' hrany incidentní s u nebo v
7 **return** C



Vrcholové pokrytí grafu

APPROX-VERTEX-COVER(G)

1 $C \leftarrow \emptyset$
2 $E' \leftarrow E$
3 **while** $E' \neq \emptyset$
4 **do** vyber libovolnou hranu (u, v) z E'
5 $C \leftarrow C \cup \{u, v\}$
6 odstraň z E' hrany incidentní s u nebo v
7 **return** C



Vrcholové pokrytí grafu

Tvrzení

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

Důkaz: Označme A množinu všech hran vybraných v kroku 4. Jakékoliv vrcholové pokrytí musí obsahovat alespoň jeden z koncových vrcholů každé hrany z A.

Pokud je C^* minimální vrcholové pokrytí, pak $|C^*| \geq |A|$.

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

Je zjevné, že APPROX-VERTEX-COVER má polynomiální časovou složitost.

Problém obchodního cestujícího

V **problému obchodního cestujícího** (traveling-salesman problem) je vstupem úplný neorientovaný graf s ohodnocením hran.

Úkolem je najít co nejkratší okružní cestu, která prochází každým vrcholem právě jednou.
(Délku cesty počítáme jako součet ohodnocení hran na této cestě.)

Problém obchodního cestujícího je NP-úplný.

(Jednoduchá redukce z problému Hamiltonovské kružnice.)

Poznámka: V problému Hamiltonovské kružnice je vstupem neorientovaný graf G a otázka je, zda v grafu G existuje cyklus procházející každým vrcholem právě jednou.

Problém obchodního cestujícího

Věta

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

Důkaz: Předpokládejme, že by pro nějaké ρ 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 $\mathsf{P} = \mathsf{NP}$.

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

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

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, ρ-aproximační algoritmus musí nějakou takovou kružnici vrátit jako výslednou okružní cestu.

Problém obchodního cestujícího

Uvažujme nyní variantu TSP, kde musí každá trojice vrcholů u, v, w splňovat **trojúhelníkovou nerovnost**

$$c(u, w) \leq c(u, v) + c(v, w) .$$

V praxi je tato podmínka často splněna.

Problém obchodního cestujícího zůstává NP-úplný i za této podmínky.

Existuje však pro něj polynomiální 1.5-aproximační algoritmus.

My si ukážeme jednodušší 2-aproximační algoritmus.

Problém obchodního cestujícího

Předpokládejme, že instancí TSP je graf G, a že optimálním řešením (které neznáme) je cesta H^*.

- Nejprve najdeme v grafu G minimální kostru T.

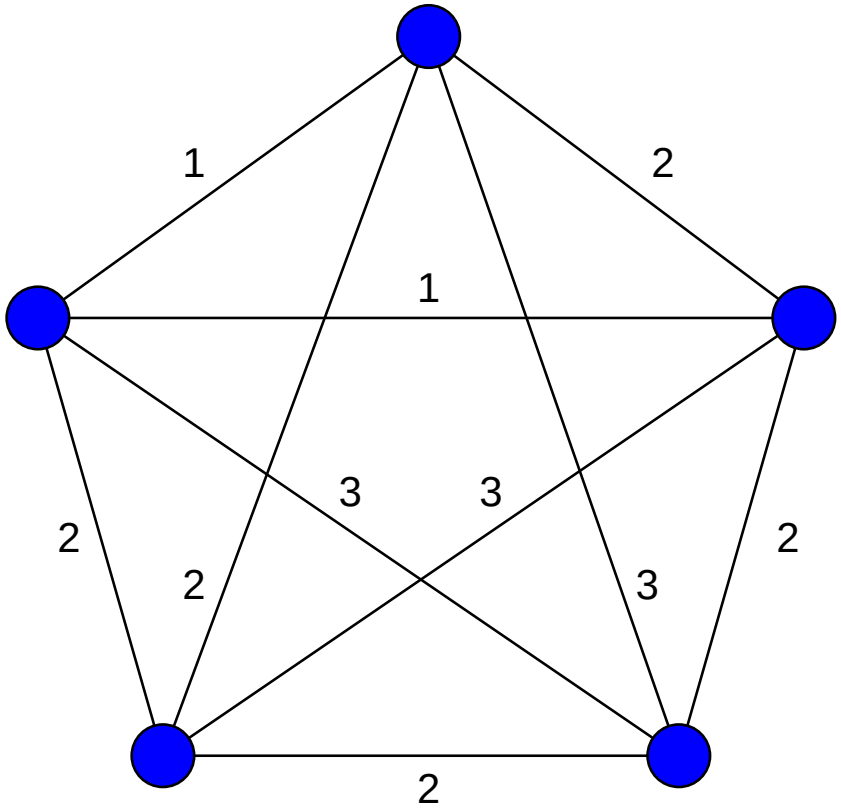
  Poznámka: Zjevně platí $c(T) \leq c(H^*)$, neboť odstraněním libovolné hrany z H^* vznikne kostra grafu G.

- Vytvoříme okružní cestu W, která odpovídá průchodu stromem T do hloubky.

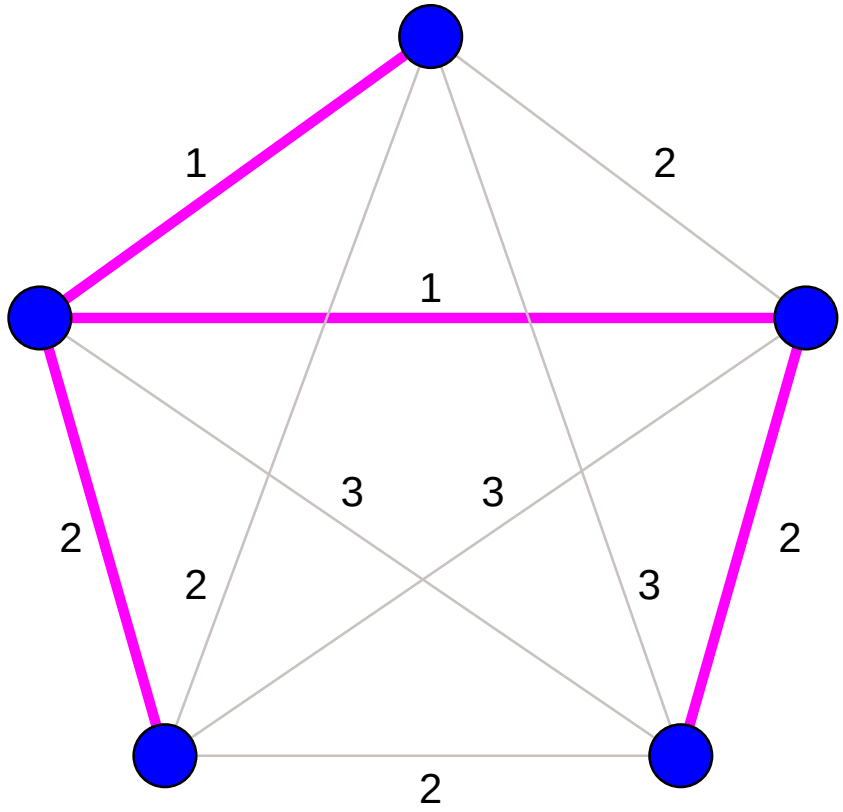
  Poznámka: Každou hranu T procházíme dvakrát. Platí tedy $c(W) = 2 \cdot c(T) \leq 2 \cdot |H^*|$.

- Z cesty W vypustíme vrcholy, které jsme již navštívili. Díky trojúhelníkové nerovnosti se tím může délka cesty pouze snížit.

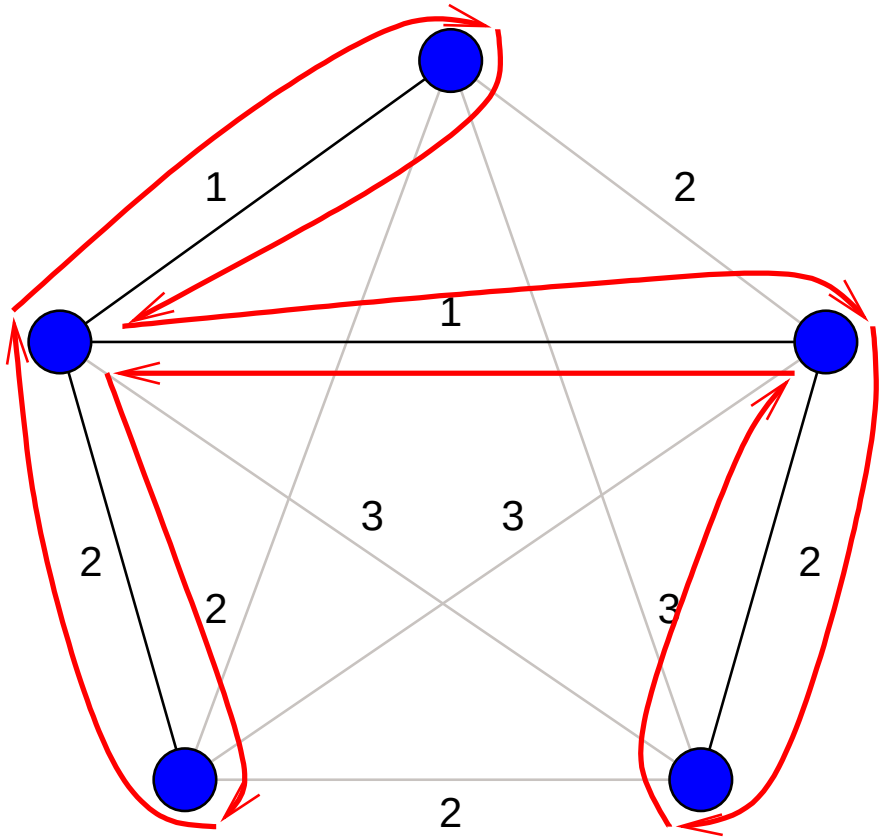
Problém obchodního cestujícího



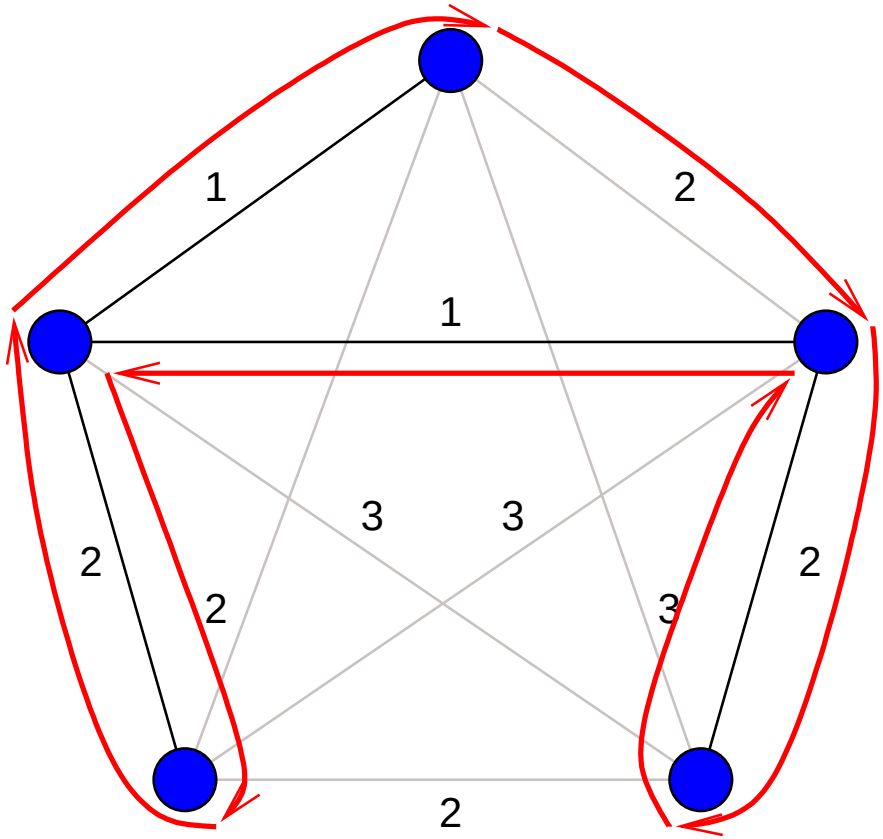
Problém obchodního cestujícího



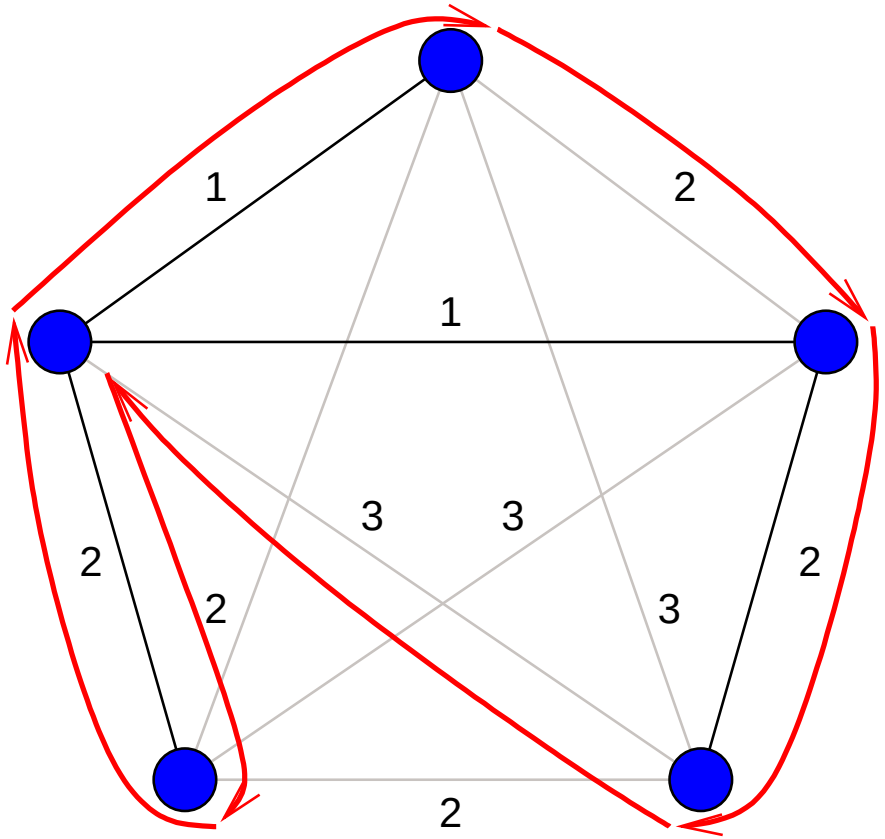
Problém obchodního cestujícího



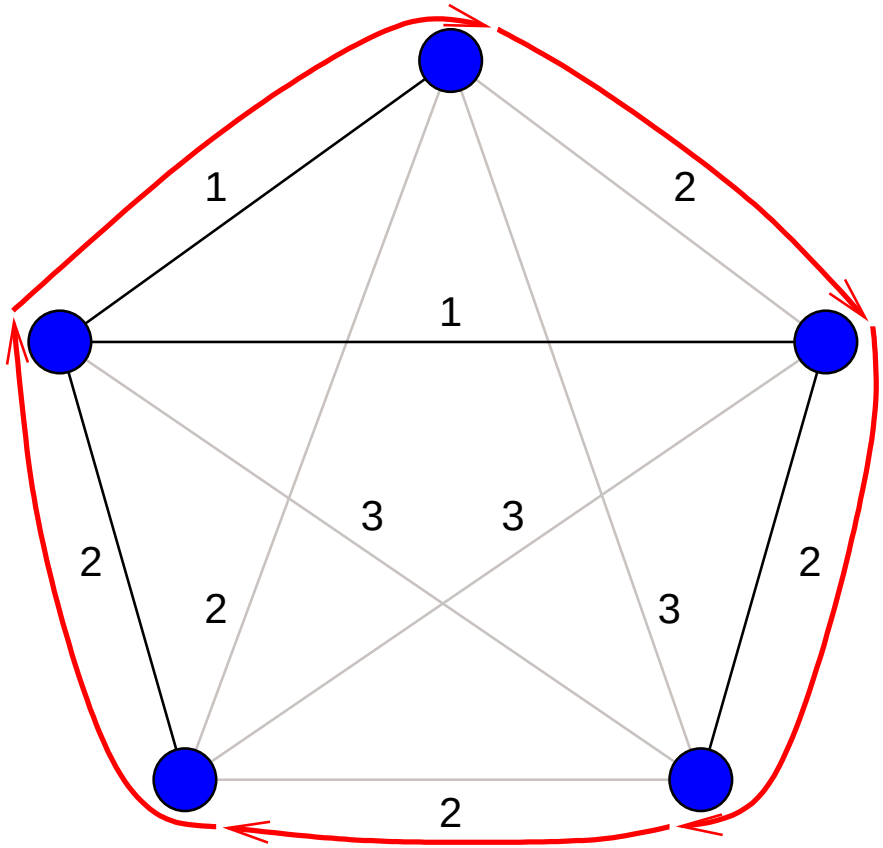
Problém obchodního cestujícího



Problém obchodního cestujícího



Problém obchodního cestujícího



Problém obchodního cestujícího