8. Převody problémů a NP-úplnost

Všechny úlohy, které jsme zatím potkali, jsme uměli vyřešit algoritmem s polynomiální časovou složitostí. V prvním přiblížení můžeme říci, že polynomialita docela dobře vystihuje praktickou použitelnost algoritmu.⟨1⟩

Existují tedy polynomiální algoritmy pro všechny úlohy? Zajisté ne, jsou dokonce i takové úlohy, jež žádným algoritmem vyřešit nelze. Ale i mezi těmi algoritmicky řešitelnými potkáme spoustu úloh, pro které zatím žádný polynomiální algoritmus není známý (ale ani neumíme dokázat, že neexistuje). Takové úlohy jsou překvapivě časté, proto se na této přednášce podíváme na několik typických příkladů.

Navíc uvidíme, že ačkoliv tyto úlohy neumíme efektivně řešit, jde mezi nimi nalézt zajímavé vztahy a pomocí těchto vztahů obtížnost problémů vzájemně porovnávat. Z těchto úvah vyrůstá i skutečná teorie složitosti se svými hierarchiemi složitostních tříd. Můžete tedy následující kapitolu považovat za malou ochutnávku toho, jak lze k třídám problémů přistupovat.

Rozhodovací problémy a převody mezi nimi

Aby se nám teorie příliš nerozkošatila, omezíme své úvahy na rozhodovací problémy. To jsou úlohy, jejichž výstupem je jediný bit – máme tedy rozhodnout, zda vstup má či nemá určenou vlastnost. Vstup přitom budeme reprezentovat řetězcem nul a jedniček – libovolnou jinou „rozumnou“ reprezentaci dokážeme na tyto řetězce převést v polynomiálním čase. Formálněji:

Definice: *Rozhodovací problém* (zkráceně *problém*) je funkce z množiny $\{0,1\}^*$ všech řetězců nad binární abecedou do množiny $\{0,1\}$.⟨2⟩

Příklad problému: *Bipartitní párování* – je dán bipartitní graf a číslo $k \in \mathbb{N}$. Zapíšeme je pomocí řetězce bitů: graf třeba maticí sousednosti, číslo dvojkově (detaily kódování budeme nadále vynechávat). Máme odpovědět, zda v zadaném grafu existuje párování, které obsahuje alespoň k hran.

(Otázka, zda existuje párování o právě k hranách, je ekvivalentní, protože můžeme libovolnou hranu z párování vypustit a bude to stále párování.)

Odbočka: Často nás samozřejmě nejen zajímá, zda párování existuje, ale také chceme nějaké konkrétní najít. I to jde pomocí rozhodovací verze problému snadno zařídit. Podobný postup funguje pro mnoho dalších problémů.

Mějme černou skříňku (fungující v polynomiálním čase), která odpoví, zda daný graf má nebo nemá párování o k hranách. Odebereme z grafu libovolnou hranu a zeptáme se, jestli i tento nový graf má párovaní velikosti k. Když má, pak tato hrana nebyla pro existenci párování potřebná, a tak ji odstraníme. Když naopak nemá

⟨1⟩ Jistě vás napadne spousta protipříkladů, jako třeba algoritmus se složitostí $\mathcal{O}(1.001^n)$, který nejspíš je použitelný, ačkoliv není polynomiální, a jiný se složitostí $\mathcal{O}(n^{100})$, u kterého je tomu naopak. Ukazuje se, že tyto případy jsou velmi řídké, takže u většiny problémů náš zjednodušený pohled funguje překvapivě dobře.

⟨2⟩ Ekvivalentně bychom se na problém mohli také dívat jako na nějakou množinu $A \subseteq \{0,1\}^*$ vstupů, na něž je odpověď 1. Tento přístup mají rádi v teorii automatů.

(hrana patří do každého párování požadované velikosti), tak si danou hranu poznamenáme a odebereme nejen ji a její vrcholy, ale také hrany, které do těchto vrcholů vedly. Toto je korektní krok, protože v původním grafu tyto vrcholy byly navzájem spárované, a tedy nemohou být spárované s žádnými jinými vrcholy. Na nový graf aplikujeme znovu tentýž postup. Výsledkem je množina hran, které patří do hledaného párování. Hran, a tedy i iterací našeho algoritmu, je polynomiálně mnoho a skříňka funguje v polynomiálním čase, takže celý algoritmus je polynomiální.⟨3⟩

Zpět z odbočky: Jak párovací problém vyřešit? Věrni matfyzáckým vtipům, převedeme ho na nějaký, který už vyřešit umíme. To už ostatně umíme – na toky v sítích. Pokaždé, když se ptáme na existenci párování velikosti alespoň k v nějakém bipartitním grafu, umíme efektivně sestrojit nějakou síť a zeptat se, zda v této síti existuje tok velikosti alespoň k. Chceme tedy přeložit vstup jednoho problému na vstup jiného problému tak, aby odpověď zůstala stejná.

Takovéto převody mezi problémy můžeme definovat i obecněji:

Definice: Jsou-li A, B rozhodovací problémy, říkáme, že A lze *převést* (neboli *redukovat*) na B (píšeme $A \to B$) právě tehdy, když existuje funkce $f : \{0,1\}^* \to \{0,1\}^*$ taková, že pro všechna $x \in \{0,1\}^*$ platí $A(x) = B(f(x))$, a navíc lze funkci f spočítat v polynomiálním čase.

Pozorování: $A \to B$ také znamená, že problém B je alespoň tak těžký jako problém A. Tím myslíme, že kdykoliv umíme vyřešit B, je vyřešit A nejvýše polynomiálně obtížnější. Speciálně platí:

Lemma: Pokud $A \to B$ a B lze řešit v polynomiálním čase, pak i A lze řešit v polynomiálním čase.

Důkaz: Nechť existuje algoritmus řešící problém B v čase $\mathcal{O}(b^k)$, kde b je délka vstupu tohoto problému a k konstanta. Mějme dále funkci f převádějící A na B v čase $\mathcal{O}(a^\ell)$ pro vstup délky a. Chceme-li nyní spočítat $A(x)$ pro nějaký vstup x délky a, spočítáme nejprve $f(x)$. To bude trvat $\mathcal{O}(a^\ell)$ a vyjde výstup délky taktéž $\mathcal{O}(a^\ell)$ – delší bychom v daném čase ani nestihli vypsat. Tento vstup pak předáme algoritmu pro problém B, který nad ním stráví čas $\mathcal{O}((a^\ell)^k) = \mathcal{O}(a^{k\ell})$. Celkový čas výpočtu tedy činí $\mathcal{O}(a^\ell + a^{k\ell})$, což je polynom v délce původního vstupu. ♡

Relace převoditelnosti jistým způsobem porovnává problémy podle obtížnosti. Nabízí se představa, že se jedná o uspořádání na množině všech problémů. Je tomu doopravdy tak?

Pozorování: O relaci „$\to$“ platí:

- *Je reflexivní* ($A \to A$) – úlohu můžeme převést na tutéž identickým zobrazením.
- *Je tranzitivní* ($A \to B \wedge B \to C \Rightarrow A \to C$) – pokud funkce f převádí A na B a funkce g převádí B na C, pak funkce $g \circ f$ převádí

⟨3⟩ Zde se skrývá hlavní důvod, proč informatici mají tak rádi polynomiální algoritmy. Třída všech polynomů je totiž nejmenší třídou funkcí, která obsahuje všechny „základní“ funkce (konstanty, n, n^2, ...) a je uzavřená na sčítání, odčítání, násobení i skládání funkcí.

A na C. Složení dvou polynomiálně vyčíslitelných funkcí je zase polynomiálně vyčíslitelná funkce, jak už jsme zpozorovali v důkazu předchozího lemmatu.

- *Není antisymetrická* – například problémy „na vstupu je řetězec začínající nulou“ a „na vstupu je řetězec končící nulou“ lze mezi sebou převádět oběma směry.
- Existují *navzájem nepřevoditelné problémy* – třeba mezi problémy „na každý vstup odpověz 0“ a „na každý vstup odpověz 1“ nemůže existovat převod ani jedním směrem.

Relacím, které jsou reflexivní a tranzitivní, ale obecně nesplňují antisymetrii, se říká *kvaziuspořádání*. Převoditelnost je tedy částečné kvaziuspořádání na množině všech problémů.[4]

Nyní se již podíváme na příklady několika problémů, které se obecně považují za těžké. Uvidíme, že každý z nich je možné převést na všechny ostatní, takže z našeho „polynomiálního“ pohledu jsou stejně obtížné.

Problém SAT – splnitelnost (satisfiability) logických formulí v CNF

Mějme nějakou logickou formuli s proměnnými a logickými spojkami. Zajímá nás, je-li tato formule *splnitelná,* tedy zda lze za proměnné dosadit 0 a 1 tak, aby formule dala výsledek 1 (byla *splněna*).

Zaměříme se na formule ve speciálním tvaru, v takzvané *konjunktivní normální formě (CNF):*

- *formule* je složena z jednotlivých *klauzulí* oddělených spojkou $\wedge$,
- každá *klauzule* je složená z *literálů* oddělených $\vee$,
- každý *literál* je buďto proměnná nebo její negace.

Vstup problému: Formule ψ v konjunktivní normální formě.

Výstup problému: Existuje-li dosazení 0 a 1 za proměnné tak, aby $\psi(\ldots) = 1$.

Příklad: Formule $(x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (x \vee \neg y \vee z) \wedge (x \vee y \vee \neg z)$ je splnitelná, stačí nastavit například $x = y = z = 1$ (jaká jsou ostatní splňující ohodnocení?). Naproti tomu formule $(x \vee y) \wedge (x \vee \neg y) \wedge \neg x$ splnitelná není, což snadno ověříme třeba vyzkoušením všech čtyř možných ohodnocení.

Poznámka: Co kdybychom chtěli zjistit, zda je splnitelná nějaká formule, která není v CNF? V logice se dokazuje, že ke každé formuli lze najít ekvivalentní formuli v CNF, ale při tom se bohužel formule může až exponenciálně prodloužit. Později ukážeme, že pro každou formuli χ existuje nějaká formule χ' v CNF, která je splnitelná právě tehdy, když je χ splnitelná. Formule χ' přitom bude dlouhá $\mathcal{O}(|\chi|)$, ale budou v ní nějaké nové proměnné.

[4] Kdybychom z něj chtěli vyrobit opravdové (byť částečné) uspořádání, mohli bychom definovat ekvivalenci $A \sim B \equiv A \to B \wedge B \to A$ a relaci převoditelnosti zavést jen na třídách této ekvivalence. Taková převoditelnost by už byla slabě antisymetrická. To je v matematice dost běžný trik, říká se mu *faktorizace* kvaziuspořádání.

Problém 3-SAT – splnitelnost formulí s krátkými klauzulemi

Pro SAT zatím není známý žádný polynomiální algoritmus. Co kdybychom zkusili problém trochu zjednodušit a uvažovat pouze formule ve speciálním tvaru?

Povolíme tedy na vstupu pouze takové formule v CNF, jejichž každá klauzule obsahuje nejvýše tři literály. Ukážeme, že tento problém je stejně těžký jako původní SAT.

Převod 3-SATu na SAT: Jelikož 3-SAT je speciálním případem SATu, poslouží tu jako převodní funkce identické zobrazení.

Převod SATu na 3-SAT: Nechť se ve formuli vyskytuje nějaká „špatná" klauzule o $k > 3$ literálech. Můžeme ji zapsat ve tvaru $(\alpha \vee \beta)$, kde α obsahuje 2 literály a β $k-2$ literálů. Pořídíme si novou proměnnou x a klauzuli nahradíme dvěma novými $(\alpha \vee x)$ a $(\beta \vee \neg x)$. První z nich obsahuje 3 literály, tedy je dobrá. Druhá má $k-1$ literálů, takže může být stále špatná, ale aspoň je kratší, takže můžeme postup opakovat.

Takto postupně nahradíme všechny špatné klauzule dobrými, což bude trvat nejvýše polynomiálně dlouho: klauzuli délky k rozebereme po $k-3$ krocích, špatných klauzulí je lineárně s délkou formule.

Zbývá ukázat, že nová formule je splnitelná právě tehdy, byla-li splnitelná formule původní. K tomu stačí ukázat, že každý jednotlivý krok převodu splnitelnost zachovává.

Pokud původní formule byla splnitelná, uvažme nějaké splňující ohodnocení proměnných. Ukážeme, že vždy můžeme novou proměnnou x nastavit tak, aby vzniklo splňující ohodnocení nové formule. Víme, že klauzule $(\alpha \vee \beta)$ byla splněna. Proto v daném ohodnocení:

- Buďto $\alpha = 1$. Pak položíme $x = 0$, takže $(\alpha \vee x)$ bude splněna díky α a $(\beta \vee \neg x)$ díky x.
- Anebo $\alpha = 0$, a tedy $\beta = 1$. Pak položíme $x = 1$, čímž bude $(\alpha \vee x)$ splněna díky x, zatímco $(\beta \vee \neg x)$ díky β.

Ostatní klauzule budou stále splněny.

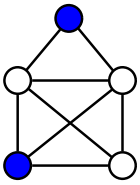
V opačném směru: pokud dostaneme splňující ohodnocení nové formule, umíme z něj získat splňující ohodnocení formule původní. Ukážeme, že stačí zapomenout proměnnou x. Všechny klauzule, kterých se naše transformace netýká, jsou nadále splněné. Co klauzule $(\alpha \vee \beta)$?

- Buďto $x = 0$, pak musí být $(\alpha \vee x)$ splněna díky α, takže $(\alpha \vee \beta)$ je také splněna díky α.
- Anebo $x = 1$, pak musí být $(\beta \vee \neg x)$ splněna díky β, takže i $(\alpha \vee \beta)$ je splněna.

Tím je převod hotov, SAT a 3-SAT jsou tedy ekvivalentní.

Problém NzMna – nezávislá množina vrcholů v grafu

Definice: Množina vrcholů grafu je *nezávislá,* pokud žádné dva vrcholy ležící v této množině nejsou spojeny hranou. (Jinými slovy nezávislá množina indukuje podgraf bez hran.)



Příklad nezávislé množiny

Na samotnou existenci nezávislé množiny se nemá smysl ptát – prázdná množina či libovolný jeden vrchol jsou vždy nezávislé. Zajímavé ale je, jestli graf obsahuje dostatečně velkou nezávislou množinu.

Vstup problému: Neorientovaný graf G a číslo $k \in \mathbb{N}$.

Výstup problému: Zda existuje nezávislá množina $A \subseteq V(G)$ velikosti alespoň k.

Převod 3-SAT → NzMna: Dostaneme formuli a máme vytvořit graf, v němž se bude nezávislá množina určené velikosti nacházet právě tehdy, je-li formule splnitelná. Myšlenka převodu bude jednoduchá: z každé klauzule budeme chtít vybrat jeden literál, jehož nastavením klauzuli splníme. Samozřejmě si musíme dát pozor, abychom v různých klauzulích nevybírali konfliktně, tj. jednou x a podruhé $\neg x$.

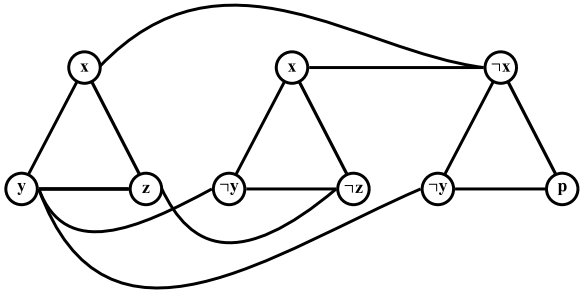
Jak to přesně zařídit: pro každou z k klauzulí zadané formule vytvoříme trojúhelník a jeho vrcholům přiřadíme literály klauzule. (Pokud by klauzule obsahovala méně literálů, prostě některé vrcholy trojúhelníka smažeme.) Navíc spojíme hranami všechny dvojice konfliktních literálů (x a $\neg x$) z různých trojúhelníků.

V tomto grafu se budeme ptát po nezávislé množině velikosti alespoň k. Jelikož z každého trojúhelníka můžeme do nezávislé množiny vybrat nejvýše jeden vrchol, jediná možnost, jak dosáhnout požadované velikosti, je vybrat z každého právě jeden vrchol. Ukážeme, že taková nezávislá množina existuje právě tehdy, je-li formule splnitelná.

Máme-li splňující ohodnocení formule, můžeme z každé klauzule vybrat jeden splněný literál. Do nezávislé množiny umístíme vrcholy odpovídající těmto literálům. Je jich právě k. Jelikož každé dva vybrané vrcholy leží v různých trojúhelnících a nikdy nemůže být splněný současně literál a jeho negace, množina je opravdu nezávislá.

A opačně: Kdykoliv dostaneme nezávislou množinu velikosti k, vybereme literály odpovídající vybraným vrcholům a příslušné proměnné nastavíme tak, abychom tyto literály splnili. Díky hranám mezi konfliktními literály se nikdy nestane, že bychom potřebovali proměnnou nastavit současně na 0 a na 1. Zbývající proměnné ohodnotíme libovolně. Jelikož jsme v každé klauzuli splnili alespoň jeden literál, jsou splněny všechny klauzule, a tedy i celá formule.

Převod je tedy korektní, zbývá rozmyslet, že běží v polynomiálním čase: Počet vrcholů grafu odpovídá počtu literálů ve formuli, počet hran je maximálně kvadratický. Každý vrchol i hranu přitom sestrojíme v polynomiálním čase, takže celý převod je také polynomiální.



Graf pro formuli $(x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z) \wedge (\neg x \vee \neg y \vee p)$

Převod NzMna → SAT: Dostaneme graf a číslo k, chceme vytvořit formuli, která je splnitelná právě tehdy, pokud se v grafu nachází nezávislá množina o alespoň k vrcholech. Tuto formuli sestrojíme následovně.

Vrcholy grafu očíslujeme od 1 do n a pořídíme si pro ně proměnné $v_1, \ldots, v_n$, které budou indikovat, zda byl příslušný vrchol vybrán do nezávislé množiny (příslušné ohodnocení proměnných tedy bude odpovídat charakteristické funkci nezávislé množiny).

Aby množina byla opravdu nezávislá, pro každou hranu $ij \in E(G)$ přidáme klauzuli $(\neg v_i \vee \neg v_j)$.

Ještě potřebujeme zkontrolovat, že množina je dostatečně velká. To neumíme provést přímo, ale použijeme lest: vyrobíme matici proměnných X tvaru $k \times n$, která bude popisovat očíslování vrcholů nezávislé množiny čísly od 1 do k. Konkrétně x_{ij} bude říkat, že v pořadí i-tý prvek nezávislé množiny je vrchol j. K tomu potřebujeme zařídit:

- Aby v každém sloupci byla nejvýše jedna jednička. Na to si pořídíme klauzule $(x_{i,j} \Rightarrow \neg x_{i',j})$ pro $i' \neq i$. (Jsou to implikace, ale můžeme je zapsat i jako disjunkce, protože $a \Rightarrow b$ je totéž jako $\neg a \vee b$.)
- Aby v každém řádku ležela právě jedna jednička. Nejprve zajistíme nejvýše jednu klauzulemi $(x_{i,j} \Rightarrow \neg x_{i,j'})$ pro $j' \neq j$. Pak přidáme klauzule $(x_{i,1} \vee x_{i,2} \vee \ldots \vee x_{i,n})$, které požadují alespoň jednu jedničku v řádku.
- Vztah mezi očíslováním a nezávislou množinou: přidáme klauzule $x_{i,j} \Rightarrow v_j$. (Všimněte si, že nezávislá množina může obsahovat i neočíslované prvky, ale to nám nevadí. Důležité je, aby jich měla k očíslovaných.)

Správnost převodu je zřejmá, ověřme ještě, že probíhá v polynomiálním čase. To plyne z toho, že vytvoříme polynomiálně mnoho klauzulí a každou z nich stihneme vypsat v lineárním čase.

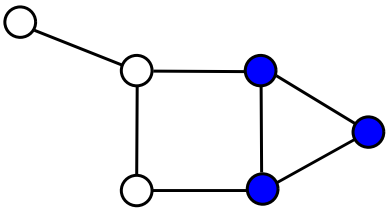
Dokázali jsme tedy, že testování existence nezávislé množiny je stejně těžké jako testování splnitelnosti formule. Pojďme se podívat na další problémy.

Problém Klika – úplný podgraf

Podobně jako nezávislou množinu můžeme v grafu hledat i *kliku* – úplný podgraf dané velikosti.

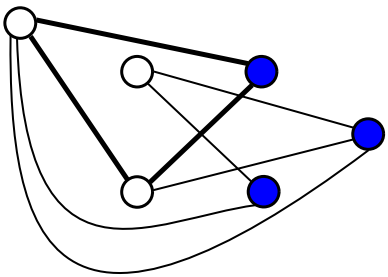
Vstup problému: Graf G a číslo $k \in N$.

Výstup problému: Existuje-li úplný podgraf grafu G na alespoň k vrcholech.



Příklad kliky

Tento problém je ekvivalentní s hledáním nezávislé množiny. Pokud v grafu prohodíme hrany a nehrany, stane se z každé kliky nezávislá množina a naopak. Převodní funkce tedy zneguje hrany a ponechá číslo k.



Prohození hran a nehran

Problém 3,3-SAT – splnitelnost s malým počtem výskytů

Než se pustíme do dalšího kombinatorického problému, předvedeme ještě jednu speciální variantu SATu, se kterou se nám bude pracovat příjemněji.

Již jsme ukázali, že SAT zůstane stejně těžký, omezíme-li se na formule s klauzulemi délky nejvýše 3. Teď budeme navíc požadovat, aby se každá proměnná vyskytovala v maximálně třech literálech. Tomuto problému se říká 3,3-SAT.

Převod 3-SATu na 3,3-SAT: Pokud se proměnná x vyskytuje v $k > 3$ literálech, nahradíme její výskyty novými proměnnými $x_1, \ldots, x_k$ a přidáme klauzule, které zabezpečí, že tyto proměnné budou vždy ohodnoceny stejně: $(x_1 \Rightarrow x_2), (x_2 \Rightarrow x_3), (x_3 \Rightarrow x_4), \ldots, (x_{k-1} \Rightarrow x_k), (x_k \Rightarrow x_1)$.

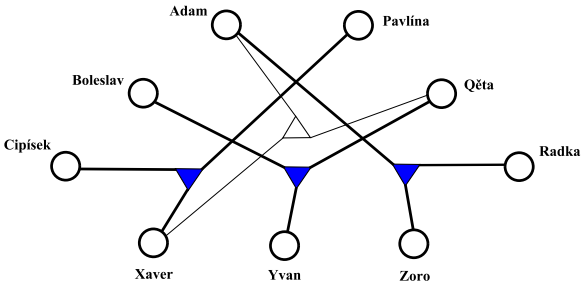
Zesílení: Můžeme dokonce zařídit, aby se každý literál vyskytoval nejvýše dvakrát (tedy že každá proměnná se vyskytuje alespoň jednou pozitivně a alespoň jednou

negativně). Pokud by se nějaká proměnná objevila ve třech stejných literálech, můžeme na ni také použít náš trik a nahradit ji třemi proměnnými. V nových klauzulích se pak bude vyskytovat jak pozitivně, tak negativně (opět připomínáme, že $a \Rightarrow b$ je jen zkratka za $\neg a \vee b$).

Problém 3D-párování

Vstup problému: Tři množiny, např. K (kluci), H (holky), Z (zvířátka) a množina $T \subseteq K \times H \times Z$ kompatibilních trojic (těch, kteří se spolu snesou).

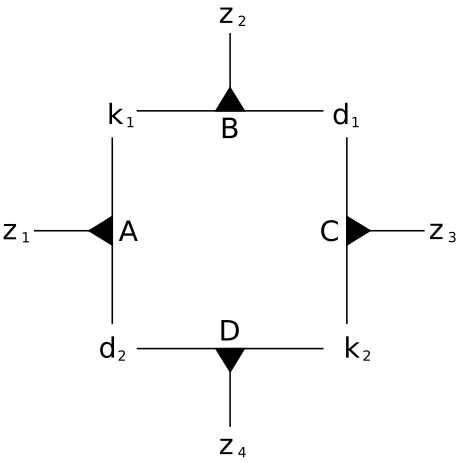
Výstup problému: Zda existuje perfektní podmnožina trojic, tedy taková, v níž se každý prvek množin K, H a Z účastní právě jedné trojice.



Ukázka 3D-párování

Převod 3,3-SATu na 3D-párování: Uvažujme trochu obecněji. Pokud chceme ukázat, že se na nějaký problém dá převést SAT, potřebujeme obvykle dvě věci: Jednak konstrukci, která bude simulovat proměnné, tedy něco, co nabývá dvou stavů *true*/*false*. Pak také potřebujeme cosi, co umí zařídit, aby každá klauzule byla splněna alespoň jednou proměnnou. Jak to provést u 3D-párování?

Uvažujme následující konfiguraci:

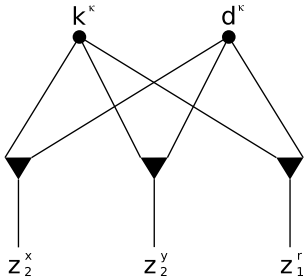


V ní se nacházejí 4 zvířátka (z_1 až z_4), 2 kluci (k_1 a k_2), 2 dívky (d_1 a d_2) a 4 trojice (A, B, C a D). Zatímco zvířátka se budou moci účastnit i jiných trojic, kluky a děvčata nikam jinam nezapojíme.

Všimneme si, že existují právě dvě možnosti, jak tuto konfiguraci spárovat. Abychom spárovali kluka k_1, tak musíme vybrat buď trojici A nebo B. Pokud si vybereme A, k_1 i d_2 už jsou spárovaní, takže si nesmíme vybrat B ani D. Pak jediná možnost, jak spárovat d_1 a k_2, je použít C. Naopak začneme-li trojicí B, vyloučíme A a D a použijeme D (situace je symetrická).

Vždy si tedy musíme vybrat dvě protější trojice v obrázku a druhé dvě nechat nevyužité. Tyto možnosti budeme používat k reprezentaci proměnných. Pro každou proměnnou si pořídíme jednu kopii obrázku. Volba $A + C$ bude odpovídat nule a nespáruje zvířátka z_2 a z_4. Volba $B + D$ reprezentuje jedničku a nespáruje z_1 a z_3. Přes tato nespárovaná zvířátka můžeme předávat informaci o hodnotě proměnné do klauzulí.

Zbývá vymyslet, jak reprezentovat klauzule. Mějme klauzuli tvaru řekněme $(x \vee y \vee \neg r)$. Potřebujeme zajistit, aby x bylo nastavené na 1 nebo y bylo nastavené na 1 nebo r na 0.



Pro takovouto klauzuli si pořídíme novou dvojici kluk a dívka, kteří budou figurovat ve třech trojicích se třemi různými zvířátky, což budou volná zvířátka z obrázků pro příslušné proměnné. Zvolíme je tak, aby byla volná při správném nastavení proměnné. Dokážeme přitom zařídit, že každé zvířátko bude použité v maximálně jedné klauzuli, neboť každý literál se vyskytuje nejvýše dvakrát a máme pro něj dvě volná zvířátka.

Ještě nám určitě zbude $2p - k$ zvířátek, kde p je počet proměnných a k počet klauzulí. Každá proměnná totiž dodá 2 volná zvířátka a každá klauzule použije jedno z nich. Přidáme proto ještě $2p - k$ párů lidí, kteří milují úplně všechna zvířátka; ti vytvoří zbývající trojice.

Snadno ověříme, že celý převod pracuje v polynomiálním čase, rozmysleme si ještě, že je korektní.

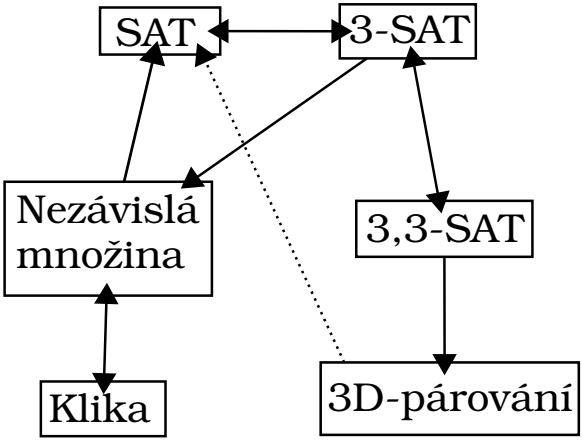
Pokud formule byla splnitelná, z každého splňujícího ohodnocení můžeme vyrobit párování v naší konstrukcí. Obrázek pro každou proměnnou spárujeme podle ohodnocení (buď $A + C$ nebo $B + D$). Pro každou klauzuli si vybereme trojici, která odpovídá některému z literálů, jimiž je klauzule splněna.

A opačně: Když nám někdo dá párovaní v naší konstrukci, dokážeme z něj vyrobit splňující ohodnocení dané formule. Podíváme se, v jakém stavu je proměnná,

a to je všechno. Z toho, že jsou správně spárované klauzule, už okamžitě víme, že jsou všechny splněné.

Ukázali jsme tedy, že na 3D-párování lze převést 3,3-SAT, a tedy i obecný SAT. Převod v opačném směru ponecháme jako cvičení, můžete ho provést podobně, jako jsme na SAT převáděli nezávislou množinu.

Jak je vidět z následujícího schématu, ukázali jsme, že všechny problémy, které jsme zkoumali, jsou navzájem převoditelné.



Problémy a převody mezi nimi

NP-úplné problémy

Všechny problémy, které jsme zatím zkoumali, měly jednu společnou vlastnost. Šlo v nich totiž o to, zda existuje nějaký objekt. Například splňující ohodnocení formule nebo klika v grafu. Kdykoliv nám přitom někdo takový objekt ukáže, umíme snadno ověřit, že má požadovanou vlastnost. Ovšem najít ho už tak snadné není. Podobně se chovají i mnohé další „vyhledávací problémy“, pojďme se na ně tedy podívat obecněji.

Definice: P je třída[⟨5⟩] rozhodovacích problémů, které jsou řešitelné v polynomiálním čase. Jinak řečeno, problém L leží v **P** právě tehdy, když existuje nějaký algoritmus A a polynom f takové, že pro každý vstup x algoritmus A doběhne v čase nejvýše $f(|x|)$ a vydá výsledek $A(x) = L(x)$.

Třída **P** tedy zachycuje naši představu o efektivně řešitelných problémech. Nyní definujeme třídu **NP**, která bude odpovídat naší představě vyhledávacích problémů.

Definice: NP je třída rozhodovacích problémů, v níž problém L leží právě tehdy, pokud existuje nějaký problém $K \in \mathbf{P}$ a polynom g, přičemž pro každý vstup x

[⟨5⟩] Formálně vzato je to množina, ale v teorii složitosti se pro množiny problémů vžil název *třídy*.

je $L(x) = 1$ právě tehdy, pokud pro nějaký řetězec y délky nejvýše $g(|x|)$ platí $K(x, y) = 1$.[6]

Co to znamená? Algoritmus K řeší problém L, ale kromě vstupu x má k dispozici ještě polynomiálně dlouhou *nápovědu* y. Přitom má platit, že je-li $L(x) = 1$, musí existovat alespoň jedna nápověda, kterou algoritmus K schválí. Pokud ovšem $L(x) = 0$, nesmí ho přesvědčit žádná nápověda.

Jinými slovy y je jakýsi *certifikát,* který stvrzuje kladnou odpověď, a problém K má za úkol certifikáty kontrolovat. Pro kladnou odpověď musí existovat alespoň jeden schválený certifikát, pro zápornou musí být všechny certifikáty odmítnuty.

Pozorování: Splnitelnost logických formulí je v **NP**. Stačí si totiž nechat napovědět, jak ohodnotit jednotlivé proměnné, a pak ověřit, je-li formule splněna. Nápověda je polynomiálně velká (dokonce lineárně), splnění zkontrolujeme také v lineárním čase. Podobně to můžeme to dokázat i o ostatních rozhodovacích problémech, se kterými jsme se zatím potkali.

Pozorování: Třída **P** leží uvnitř **NP**. Pokud totiž problém umíme řešit v polynomiálním čase bez nápovědy, tak to zvládneme v polynomiálním čase i s nápovědou. Algoritmus K tedy bude ignorovat nápovědy a odpověď spočítá přímo ze vstupu.

Otázka: Jsou třídy **P** a **NP** různé? Na to se teoretičtí informatici snaží odpovědět už od 70. let minulého století a postupně se z toto stal asi vůbec nejslavnější otevřený problém informatiky.

Například o žádném z našich problémů nevíme, zda se nachází v **P**. Brzy uvidíme, že to jsou v jistém smyslu nejtěžší problémy v **NP**.

Definice: Problém L nazveme **NP**-*těžký*, je-li na něj převoditelný každý problém z **NP**. Pokud navíc $L \in$ **NP**, budeme říkat, že je **NP**-*úplný.*

NP-úplné problémy jsou tedy nejtěžšími problémy v **NP**, aspoň v našem uspořádání převoditelností.

Lemma: Pokud nějaký **NP**-těžký problém L leží v **P**, pak **P** = **NP**.
Důkaz: Již víme, že **P** $\subseteq$ **NP**, takže stačí dokázat opačnou nerovnost. Vezměme libovolný problém $A \in$ **NP**. Z úplnosti problému L víme, že A lze převést na L. Ovšem problémy převoditelné na něco z **P** jsou samy také v **P**. ♡

Z definice **NP**-úplnosti ale vůbec není jasné, že nějaký **NP**-úplný problém doopravdy existuje. Odpověď je překvapivá:

Věta (Cookova): SAT je **NP**-úplný.

Důkaz této věty je značně technický a alespoň v hrubých rysech ho předvedeme v závěru teto kapitoly. Jakmile ale máme jeden **NP**-úplný problém, můžeme velice snadno dokazovat i **NP**-úplnost dalších:

Lemma: Mějme dva problémy $L, M \in$ **NP**. Pokud L je **NP**-úplný a $L \to M$, pak M je také **NP**-úplný.
Důkaz: Jelikož M leží v **NP**, stačí o něm dokázat, že je **NP**-těžký, tedy že na něj lze převést libovolný problém z **NP**. Uvažme tedy nějaký problém $Q \in$ **NP**. Jelikož L

[6] Rozhodovací problémy mají na vstupu řetězec bitů. Tak jaképak x, y? Máme samozřejmě na mysli nějaké binární kódování této dvojice.

je **NP**-úplný, musí platit $Q \to L$. Převoditelnost je ovšem tranzitivní, takže z $Q \to L$ a $L \to M$ plyne $Q \to M$. ♡

Důsledek: Cokoliv, na co jsme uměli převést SAT, je také NP-úplné. Například nezávislá množina, různé varianty SATu, klika v grafu ...

Dva možné světy: Jestli je **P** = **NP** nevíme a nejspíš ještě dlouho nebudeme vědět. Nechme se ale na chvíli unášet fantazií a zkusme si představit, jak by vypadaly světy, v nichž platí jedna nebo druhá možnost:

- **P** = **NP** – to je na první pohled idylický svět, v němž jde každý vyhledávací problém vyřešit v polynomiálním čase, nejspíš tedy i prakticky efektivně. Má to i své stinné stránky: například jsme přišli o veškeré efektivní šifrování – rozmyslete si, že pokud umíme vypočítat nějakou funkci v polynomiálním čase, umíme efektivně spočítat i její inverzi.
- **P** ≠ **NP** – tehdy jsou **P** a **NP**-úplné dvě disjunktní třídy. SAT a ostatní **NP**-úplné problémy nejsou řešitelné v polynomiálním čase. Je ale stále možné, že aspoň na některé z nich existují prakticky použitelné algoritmy, třeba o složitosti $\Theta((1+\varepsilon)^n)$ nebo $\Theta(n^{\log n/100})$. Ví se, že třída **NP** by pak obsahovala i problémy, které leží „mezi" **P** a **NP**-úplnými.

Katalog NP-úplných problémů

Pokud se setkáme s problémem, který neumíme zařadit do **P**, hodí se vyzkoušet, zda je **NP**-úplný. K tomu se hodí mít alespoň základní zásobu „učebnicových" **NP**-úplných problémů, abychom si mohli vybrat, z čeho převádět. Ukážeme tedy katalog několika nejběžnějších **NP**-úplných problémů. O některých jsme to dokázali během této kapitoly, u ostatních alespoň naznačíme, jak na ně.

Standardní NP-úplné problémy:

- *Logické:*
  - SAT (splnitelnost logických formulí v CNF)
  - 3-SAT (každá klauzule obsahuje max. 3 literály)
  - 3,3-SAT (a navíc každá proměnná se vyskytuje nejvýše 3×)
  - SAT pro obecné formule (nejen CNF; ukážeme níže)
  - Obvodový SAT (místo formule booleovský obvod; viz níže)
- *Grafové:*
  - Nezávislá množina (existuje množina alespoň k vrcholů taková, že žádné dva nejsou propojeny hranou?)
  - Klika (existuje úplný podgraf na k vrcholech?)
  - 3D-párování (tři množiny se zadanými trojicemi, existuje taková množina disjunktních trojic, ve které jsou všechny prvky právě jednou?)

- Barvení grafu (lze obarvit vrcholy k barvami tak, aby vrcholy stejné barvy nebyly nikdy spojeny hranou? **NP**-úplné už pro $k = 3$)
- Hamiltonovská cesta (cesta obsahující všechny vrcholy)
- Hamiltonovská kružnice (opět obsahuje všechny vrcholy)

- *Číselné:*
  - Batoh (nejjednodušší verze: má daná množina čísel podmnožinu s daným součtem?)
  - Batoh – optimalizace (podobně jako u předchozího problému, ale místo množiny čísel máme množinu předmětů s vahami a cenami, chceme co nejdražší podmnožinu, jejíž váha nepřesáhne zadanou kapacitu batohu)
  - Dva loupežníci (lze rozdělit danou množinu čísel na dvě podmnožiny se stejným součtem?)
  - $\mathbf{Ax} = \mathbf{b}$ (soustava celočíselných lineárních rovnic; je dána matice $\mathbf{A} \in \{0,1\}^{m \times n}$ a vektor $\mathbf{b} \in \{0,1\}^m$, existuje vektor $\mathbf{x} \in \{0,1\}^n$ takový, že $\mathbf{Ax} = \mathbf{b}$?)

Náznak důkazu Cookovy věty

Zbývá dokázat Cookovu větu. Technické detaily si odpustíme, ale aspoň načrtneme základní myšlenku důkazu.

Potřebujeme tedy dokázat, že SAT je **NP**-úplný, a to z definice. Ukážeme to ale nejprve pro jiný problém, pro takzvaný *obvodový SAT.* V něm máme na vstupu booleovský obvod (hradlovou síť) s jedním výstupem a ptáme se, zda jí můžeme přivést na vstupy takové hodnoty, aby vydala výsledek 1.

To je obecnější než SAT pro formule (dokonce i neomezíme-li formule na CNF), protože každou formuli můžeme přeložit na lineárně velký obvod. (Platí i opačně, že každému obvodu s jedním výstupem můžeme přiřadit ekvivalentní formuli, ale ta může být až exponenciálně velká.)

Nejprve tedy dokážeme **NP**-úplnost obvodového SATu a pak ho převedeme na obyčejný SAT v CNF. Tím bude důkaz Cookovy věty hotový. Začněme lemmatem, v němž bude koncentrováno vše technické.

Lemma: Nechť L je problém ležící v **P**. Potom existuje polynom p a algoritmus, který pro každé n sestrojí v čase $p(n)$ hradlovou síť B_n s n vstupy a jedním výstupem, která řeší L. Tedy pro všechny řetězce x musí platit $B_n(x) = L(x)$.

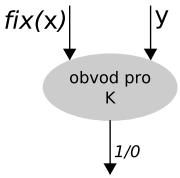
Důkaz: Náznakem. Na základě zkušeností z Principů počítačů intuitivně chápeme počítače jako nějaké složité booleovské obvody, jejichž stav se mění v čase. Uvažme tedy nějaký problém $L \in \mathbf{P}$ a polynomiální algoritmus, který ho řeší. Pro vstup velikosti n algoritmus doběhne v čase T polynomiálním v n a spotřebuje $\mathcal{O}(T)$ buněk paměti. Stačí nám tedy „počítač s pamětí velkou $\mathcal{O}(T)$“, což je nějaký booleovský obvod velikosti polynomiální v T, a tedy i v n. Vývoj v čase ošetříme tak, že sestrojíme T kopií tohoto obvodu, každá z nich bude odpovídat jednomu kroku výpočtu

a bude propojena s „minulou“ a „budoucí“ kopií. Tím sestrojíme booleovský obvod, který bude řešit problém L pro vstupy velikosti n a bude polynomiálně velký vzhledem k n.

Ekvivalentní definice NP: Pro důkaz následující věty si dovolíme drobnou úpravu v definici třídy **NP**. Budeme chtít, aby nápověda měla pevnou velikost, závislou pouze na velikosti vstupu (tedy: $|y| = g(|x|)$ namísto $|y| \leq g(|x|)$). Proč je taková úprava bez újmy na obecnosti? Stačí původní nápovědu doplnit na požadovanou délku nějakými „mezerami“, které budeme při ověřování nápovědy ignorovat.

Věta: Obvodový SAT je **NP**-úplný.
Důkaz: Obvodový SAT evidentně leží v **NP** – stačí si nechat poradit vstup, síť topologicky setřídit a v tomto pořadí počítat hodnoty hradel.

Mějme nyní nějaký problém L z **NP**, o němž chceme dokázat, že se dá převést na obvodový SAT. Když nám někdo předloží nějaký vstup x délky n, spočítáme velikost nápovědy $g(n)$. Víme, že algoritmus K, který kontroluje, zda nápověda je správně, je v **P**. Využijeme předchozí lemma, abychom získali obvod, který pro konkrétní velikost vstupu n počítá to, co kontrolní algoritmus K. Vstupem tohoto obvodu bude x (vstup problému L) a nápověda y. Na výstupu se dozvíme, zda je nápověda správná. Velikost tohoto obvodu bude činit $p(g(n))$, což je také polynom.



V tomto obvodu zafixujeme vstup x (na místa vstupu dosadíme konkrétní hodnoty z x). Tím získáme obvod, jehož vstup je jen y, a chceme zjistit, zda za y můžeme dosadit nějaké hodnoty tak, aby na výstupu bylo *true*. Jinými slovy, ptáme se, zda je tento obvod splnitelný.

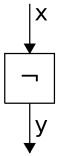
Ukázali jsme tedy, že pro libovolný problém z **NP** dokážeme sestrojit funkci, která pro každý vstup x v polynomiálním čase vytvoří obvod, jenž je splnitelný pravě tehdy, když odpověď tohoto problému na vstup x má být kladná. To je přesně převod z daného problému na obvodový SAT. ♡

Lemma: Obvodový SAT se dá převést na 3-SAT.
Důkaz: Budeme postupně budovat formuli v konjunktivní normální formě. Každý booleovský obvod se dá v polynomiálním čase převést na ekvivalentní obvod, ve kterém se vyskytují jen hradla AND a NOT, takže stačí najít klauzule odpovídající těmto hradlům. Pro každé hradlo v obvodu zavedeme novou proměnnou popisující jeho výstup. Přidáme klauzule, které nám kontrolují, že toto hradlo máme ohodnocené konzistentně.

Převod hradla NOT: Na vstupu hradla budeme mít nějakou proměnnou x (která přišla buďto přímo ze vstupu celého obvodu, nebo je to proměnná, která vznikla na výstupu nějakého hradla) a na výstupu proměnnou y. Přidáme klauzule, které nám

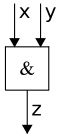
zaručí, že jedna proměnná bude negací té druhé:

$$(x \vee y),\quad (\neg x \vee \neg y).$$



Převod hradla AND: Hradlo má vstupy x, y a výstup z. Potřebujeme přidat klauzule, které nám popisují, jak se má hradlo AND chovat. Tyto vztahy přepíšeme do konjunktivní normální formy:

$$\left.\begin{array}{c} x \,\&\, y \Rightarrow z \\ \neg x \Rightarrow \neg z \\ \neg y \Rightarrow \neg z \end{array}\right\} \quad \begin{array}{c} (z \vee \neg x \vee \neg y) \\ (\neg z \vee x) \\ (\neg z \vee y) \end{array}$$



Tím v polynomiálním čase vytvoříme formuli, která je splnitelná právě tehdy, je-li splnitelný zadaný obvod. Ve splňujícím ohodnocení formule bude obsaženo jak splňující ohodnocení obvodu, tak výstupy všech hradel obvodu. ♡

Poznámka: Tím jsme také odpověděli na otázku, kterou jsme si kladli při zavádění SATu, tedy zda omezením na CNF o něco přijdeme. Teď už víme, že nepřijdeme – libovolná booleovská formule se dá přímočaře převést na obvod a ten zase na formuli v CNF. Zavádíme sice nové proměnné, ale nová formule je splnitelná právě tehdy, kdy ta původní.

Cvičení

1. Vrcholové pokrytí grafu je množina vrcholů, která obsahuje alespoň jeden vrchol z každé hrany. (Chceme na křižovatky rozmístit strážníky tak, aby každou ulici alespoň jeden hlídal.) Ukažte vzájemné převody mezi problém nezávislé množiny a problémem „Existuje vrcholové pokrytí velikosti nejvýše k?“.
2. Zesilte náš převod SATu na nezávislou množinu tak, aby vytvářel grafy s maximálním stupněm 4.
3. Dokažte **NP**-úplnost problému $\mathbf{Ax} = \mathbf{b}$ z katalogu.
4.* Dokažte **NP**-úplnost problému barvení grafu z katalogu.
5. Ukažte, že barvení grafu jednou nebo dvěma barvami je snadné.
6. Převeďte batoh na dva loupežníky a opačně.
7. Dokažte **NP**-úplnost problému batohu.
8. Pokud bychom definovali **P**-úplnost analogicky k **NP**-úplnosti, které problémy z **P** by byly **P**-úplné?
9.* Dokažte lemma o vztahu mezi problémy z **P** a hradlovými sítěmi pomocí výpočetního modelu RAM.