11. NP-úplné problémy *(zapsali F. Kačmarik, R. Krivák, D. Remiš)*

Dosud jsme zkoumali problémy, které se nás ptaly na to, jestli něco existuje. Například jsme dostali formuli a problém splnitelnosti se nás ptal, zda existuje ohodnocení proměnných takové, že formule platí. Nebo v případe nezávislých množin jsme dostali graf a číslo k a ptali jsme se, jestli v grafu existuje nezávislá množina, která obsahuje alespoň k vrcholů. Tyto otázky měly společné to, že když nám někdo zadal nějaký objekt, uměli jsme efektivně říci, zda je to objekt, který hledáme. Například pokud dostaneme ohodnocení proměnných logické formule, stačí jen dosadit a spočítat, že formule je *true* nebo *false*. Zjistit, že nějaký objekt je ten, který hledáme, umíme efektivně. Těžké na tom je takový objekt najít. Což vede k definici obecných vyhledávacích problémů, kterým se říká třída problémů NP. Definujeme si ji pořádně, ale nejdříve začneme trošičku jednodušší třídou.

Definice: P je *třída rozhodovacích problémů*, které jsou řešitelné v polynomiálním čase. Jinak řečeno, problém $L \in P \Leftrightarrow \exists$ polynom f a $\exists$ algoritmus A takový, že $\forall x : L(x) = A(x)$ a $A(x)$ doběhne v čase $\mathcal{O}(f(x))$.

Třída P odpovídá tomu, o čem jsme se shodli, že umíme efektivně řešit. Nadefinujme tedy třídu NP:

Definice: NP je *třída rozhodovacích problémů* takových, že $L \in$ NP právě tehdy, když $\exists$ problém $K \in$ P a $\exists$ polynom g takový, že pro $\forall x$ platí $L(x) = 1 \Leftrightarrow \exists$ nápověda $y : |y| \leq g(|x|)$ a současně $K(x, y) = 1$.

Pozorování: Splnitelnost logických formulí je v NP. Stačí si totiž nechat napovědět, jak ohodnotit jednotlivé proměnné a pak ověřit, jestli je formule splněna. Nápověda je polynomiálně velká (dokonce lineárně), splnění zkontrolujeme také v lineárním čase. Odpovíme tedy ano právě tehdy, existuje-li nápověda, která nás přesvědčí, tedy pokud je formule splnitelná.

Pozorování: Třída P leží uvnitř NP. V podstatě říkáme, že když máme problém, který umíme řešit v polynomiálním čase bez nápovědy, tak to zvládneme v polynomiálním čase i s nápovědou.

Nasbírali jsme problémy, které jsou v NP, ale nevíme, jestli jsou v P. Brzy ukážeme, že to jsou v jistém smyslu nejtěžší problémy v NP. Nadefinujme si:

Definice: Problém L je NP-*těžký* právě tehdy, když je na něj převoditelný každý problém z NP. (Viz definici převodů z minulé přednášky)

Také platí, že pokud umíme řešit nějaký NP-těžký problém v polynomiálním čase, pak umíme vyřešit v polynomiálním čase vše v NP, a tedy P = NP. (To už také víme z minulé přednášky.)

My se budeme zabývat problémy, které jsou NP-těžké a samotné jsou v NP. Takovým problémům se říká NP-úplné.

Definice: Problém L je NP-*úplný* právě tehdy, když L je NP-těžký a $L \in$ NP.

NP-úplné problémy jsou tedy ve své podstatě nejtěžší problémy, které leží v NP. Kdybychom uměli vyřešit nějaký NP-úplný problém v polynomiálním čase, pak všechno v NP je řešitelné v polynomiálním čase. Bohužel to, jestli nějaký

NP-úplný problém lze řešit v polynomiálním čase, se neví. Otázka, jestli P = NP, je asi nejznámější otevřený problém v celé teoretické informatice.

Kde ale nějaký NP-úplný problém vzít? K tomu se nám bude velice hodit následující věta:

Věta (Cookova): SAT je NP-úplný.

Důkaz je značně technický, přibližně ho naznačíme později. Přímým důsledkem je, že cokoli v NP je převoditelné na SAT. K dokazování NP-úplnosti dalších problémů použijeme následující větičku:

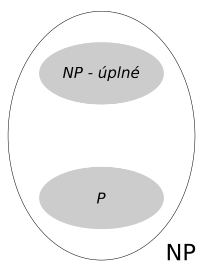
Větička: Pokud problém L je NP-úplný a L se dá převést na $M \in$ NP ($L \rightarrow M$), pak M je také NP-úplný.

Důkaz: Tuto větičku stačí dokázat pro NP-těžkost, NP-úplnost plyne okamžitě z toho, že problémy jsou NP-těžké a leží v NP (podle předpokladu).

Víme, že L se dá převést na M nějakou funkcí f. Jelikož L je NP-úplný, pak pro každý problém $Q \in$ NP existuje nějaká funkce g, která převede Q na L. Stačí tedy složit funkci f s funkcí g, čímž získáme funkci pracující opět v polynomiálním čase, která převede Q na M. Každý problém z NP se tedy dá převést na problém M. ♡

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

Jak taková třída NP vypadá? Představme si všechny problémy třídy NP, jakoby seřazené zhora nadolu podle obtížnosti problémů, kde porovnání dvou problémů určuje převoditelnost (viz obrázek).



Struktura třídy NP

Obecně mohou nastat dvě situace. Protože nevíme, jestli P = NP. Jestli ano, pak všechno je jedna a ta samá třída. To by bylo v některých případech nepraktické, např. každá šifra by byla jednoduše rozluštitelná. Jestli ne, NP-úplné problémy určitě neleží v P, takže P a NP-úplné problémy jsou dvě disjunktní části NP. Také se dá dokázat (to dělat nebudeme, ale je dobré to vědět), že ještě něco leží mezi nimi, tedy že existuje problém, který je v NP, není v P a není NP-úplný.

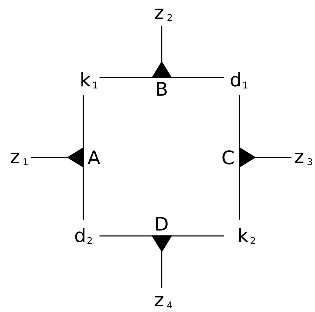
Katalog NP-úplných problémů

Ukážeme si několik základních NP-úplných problémů. O některých jsme to dokázali na minulé přednášce, o dalších si to dokážeme nyní, zbylým se na zoubek podíváme na cvičeních.

- *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 třikrát)
  - SAT pro obecné formule (nejen CNF)
  - Obvodový SAT (není to formule, ale obvod)
- *grafové:*
  - Nezávislá množina (množina alespoň k vrcholů taková, že žádné dva nejsou propojeny hranou)
  - Klika (úplný podgraf na k vrcholech)
  - 3D párování (tři množiny se zadanými trojicemi, najít takovou množinu disjunktních trojic, ve které jsou všechny prvky)
  - Barvení grafu (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 [právě jednou])
  - Hamiltonovská kružnice (kružnice, která navštíví všechny vrcholy [právě jednou])
- *číselné:*
  - Batoh (nejjednodušší verze: dána množina čísel, zjistit, zda existuje podmnožina se zadaným součtem)
  - Loupežníci (rozdělit množinu na dvě podmnožiny se stejným součtem)
  - $Ax = b$ (soustava celočíslených lineárních rovnic; x_i mohou být pouze 0 nebo 1; NP-úplné i pokud $A_{ij} \in \{0, 1\}$ a $b_i \in \{0, 1\}$)
  - Celočíselné lineární programování (existuje vektor nezáporných celočísených x takový, že $Ax \leq b$)

Převoditelnost 3,3-SAT na 3D-párování

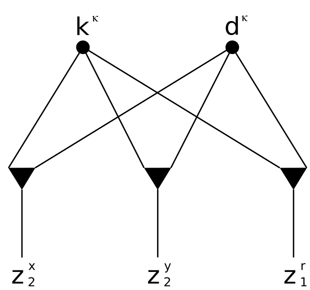
Když chceme ukázat, že na něco se dá převést SAT, potřebujeme obvykle dvě věci. Konstrukci, která bude simulovat proměnné, tedy něco, co nabývá dvou stavů *true*/*false*, a něco, co bude reprezentovat klauzule a umí zařídit, aby každá klauzule byla splněna alespoň jednou proměnnou. Jenom pro připomenutí, máme množinu kluků, děvčat, zvířátek a nějaké trojice, kdo se s kým snese, a chceme vybrat trojice tak, aby se v nich každý kluk, holka, zvířátko vyskytovalo právě jednou. Najdeme si takovouto konfiguraci:



4 zvířátka, 2 kluci, 2 dívky a takové 4 trojice, které označíme A, B, C, D. Ještě předpokládáme, že zvířátka se mohou účastnit nějakých jiných trojic, ale tito čtyři lidé se vyskytují pouze v těchto čtyřech trojicích a nikde jinde. Všimneme si, že existují právě dvě možnosti, jak tento obrázek spárovat. Abychom spárovali kluka k_1, tak si musíme vybrat A nebo B. Když 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 C. Jedna možnost je tedy vybrat si A a C a jelikož je obrázek symetrický, tak když vybereme místo A trojici B, dostaneme B a D. Vždy si tedy vybereme dvě protější trojice v obrázku.

Takovýto obrázek budeme používat k reprezentaci proměnných. Pro každou proměnnou si nakreslíme takový obrázek a to, že A bude spárované s C, bude odpovídat tomu, že $x = 1$, a spárování B a D odpovídá $x = 0$. Pokud jsme použili A a C, zvířata se sudými čísly, tj. z_2 a z_4, horní a dolní jsou nespárovaná a pokud jsme použili B a D, zvířátka z_1 a z_3 zůstala nespárovaná. Přes tyto nespárovaná zvířátka můžeme předávat informaci, jestli proměnná x má hodnotu *true* nebo *false* do dalších částí grafu.

Zbývá vymyslet, jak reprezentovat klauzule. Klauzule budou vypadat jako trojice literálů: $\kappa = (x \vee y \vee \neg r)$ Potřebujeme zajistit, aby x bylo nastavené na 1 nebo y bylo nastavené na 1 nebo r na 0.



Pro takovouto klauzuli si pořídíme dvojici kluk-dívka, kteří budou figurovat ve třech trojicích se třemi různými zvířátky, což mají být volná zvířátka z obrázků pro příslušné proměnné. A zařídíme to tak, aby každé zvířátko bylo použité maximálně v jedné takové trojici, což jde proto, že každý literál se vyskytuje maximálně dvakrát a pro každý literál máme dvě volná zvířátka, z čehož plyne, že zvířátek je dost pro všechny klauzule.

Ještě nám ale určitě zbude $2p - k$ zvířátek, kde p je počet proměnných, tak přidáme ještě $2p - k$ párů kluk-děvče, kteří milují všechna zvířátka, a ti vytvoří zbývající páry.

Pokud formule byla splnitelná, pak ze splňujícího ohodnocení můžeme vyrobit párování s naší konstrukcí. Obrázek pro každou proměnnou spárujeme podle ohodnocení, tj. proměnná je 0 nebo 1 a pro každou klauzuli si vybereme některou z proměnných, kterými je ta klauzule splněna. Funguje to také ale i opačně. Když nám někdo dá párovaní v naší konstrukci, pak z něho dokážeme 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.

Zbývá ověřit, že naše redukce funguje v polynomiálním čase. Pro každou klauzuli spotřebujeme konstantně mnoho času, $2p - k$ je také polynomiálně mnoho a když to sečteme, máme polynomiální čas vzhledem k velikosti vstupní formule. Tím je převod hotový a můžeme 3D-párování zařadit mezi NP-úplné problémy.

Náznak důkazu Cookovy věty

Abychom mohli budovat teorii NP-úplnosti, potřebujeme alespoň jeden problém, o kterém dokážeme, že je NP-úplný, z definice. Cookova věta říká o NP-úplnosti SAT-u, ale nám se to hodí dokázat o trošku jiném problému – *obvodovém SAT-u.*

Obvodový SAT je splnitelnost, která nepracuje s formulemi, ale s booleovskými obvody. Každá formule se dá přepsat do booleovského obvodu, který ji počítá, takže dává smysl zavést splnitelnost i pro obvody. Naše obvody budou mít nějaké vstupy a jenom jeden výstup. Budeme se ptát, jestli se vstupy tohoto obvodu dají nastavit tak, abychom na výstupu dostali *true*.

Nejprve dokážeme NP-úplnost *obvodového SAT-u* a pak ukážeme, že se dá převést na obyčejný SAT v CNF. Tím bude důkaz Cookovy věty hotový.

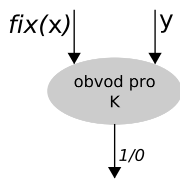
Věta: Obvodový SAT je NP-úplný.

Důkaz: Náznakem. Na základě zkušeností z Principů počítačů intuitivně chápeme počítače jako nějaké složité booleovské obvody, jejichž stav se mění v čase. Uvažme nějaký problém $L \in \mathrm{P}$ a polynomiální algoritmus, který ho řeší. Pro vstup velikosti n tedy 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.

Ještě si dovolíme drobnou úpravu v definici třídy NP. Budeme chtít, aby nápověda byla měla pevnou velikost, závislou pouze na velikosti vstupu (tedy: $|y| = g(|x|)$). Proč je taková úprava BÚNO? Jistě si dovedete představit, že původní nápovědu doplníme na požadovanou délku nějakými „mezerami“, které program ignoruje.

(Tedy upravíme program tak, aby mu nevadilo, že dostane na konci nápovědy nějak kódované mezery.)

Máme tedy nějaký problém L z NP a chceme dokázat, že L se dá převést na obvodový SAT. Když nám někdo předloží nějaký vstup x (chápeme jako vektor $(x_1, x_2, \ldots, x_n)$), spočítáme velikost nápovědy $g(|x|)$. Víme, že kontrolní algoritmus K (který kontroluje, zda nápověda je správně) je v P. Využijeme intuice o obvodech, abychom získali obvod, který pro konkrétní velikost vstupu x počítá to, co kontrolní algoritmus K. Na vstupu tohoto obvodu bude x (vstup problému L) a nápověda y. Na výstupu nám řekne, jestli je nápověda správná. Velikost vstupu tohoto obvodu bude tedy $|x| + g(|x|)$, což je polynom.



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

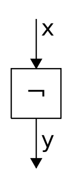
Pro libovolný problém z NP tak dokážeme sestrojit funkci, která pro každý vstup x v polynomiálním čase vytvoří obvod, který je splnitelný právě tehdy, když odpověď tohoto problému na vstup x má být *true*. Tedy libovolný problém z NP se dá v polynomiálním čase převést na obvodový SAT. ♡

Lemma: Obvodový SAT se dá převést na 3-SAT.

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

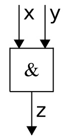
Převod hradla NOT: na vstupu hradla budeme mít nějakou proměnnou x (která přišla buďto přímo ze vstupu toho celého obvodu nebo je to proměnná, která vznikla na výstupu nějakého hradla) a na výstupu proměnnou y. Přidáme klauzule, které nám zaručí, že jedna proměnná bude negací té druhé:

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



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:

$$\left.\begin{array}{c} x \mathbin{\&} y \Rightarrow z \\ \neg x \Rightarrow \neg z \\ \neg y \Rightarrow \neg z \end{array}\right\} \qquad \begin{array}{c} (z \lor \neg x \lor \neg y) \\ (\neg z \lor x) \\ (\neg z \lor y) \end{array}$$



Když chceme převádět obvodový SAT na 3-SAT, obvod nejdříve přeložíme na takový, ve kterém jsou jen hradla AND a NOT, a pak hradla tohoto obvodu přeložíme na klauzule. Formule vzniklá z takovýchto klauzulí je splnitelná právě tehdy, když je splnitelný daný obvod. Převod pracuje v polynomiálním čase. ♡

Poznámka: Když jsme zaváděli SAT, omezili jsme se jen na formule, které jsou v konjunktivní normální formě (CNF). Teď už víme, že splnitelnost obecné booleovské formule dokážeme převést na obvodovou splnitelnost a tu pak převést na 3-SAT. Opačný převod je samozřejmě triviální, takže obecný SAT je ve skutečnosti ekvivalentní s naším „standardním“ SATem pro CNF.