9. Co si počít s těžkým problémem

V předchozí kapitole jsme zjistili, že leckteré rozhodovací problémy jsou **NP**-úplné. Z toho plyne, že jsou ekvivalentní, ale bohužel také, že ani jeden z nich zatím neumíme vyřešit v polynomiálním čase.

Často se stane, že problém, který v životě potkáme, patří mezi **NP**-úplné. Přesněji řečeno spíš než s rozhodovacím problémem se potkáme s problémem *optimalizačním,* ve kterém jde o nalezení *nejlepšího* objektu s danou vlastností. To může být třeba největší nezávislá množina v grafu nebo obarvení grafu nejmenším možným počtem barev. Kdybychom uměli efektivně řešit optimalizační problém, umíme samozřejmě řešit i příslušný rozhodovací, takže pokud $\mathbf{P} \neq \mathbf{NP}$, jsou i optimalizační problémy těžké.

Ale co naplat, svět nám takové úlohy předkládá a my je potřebujeme vyřešit. Naštěstí situace není zase tak beznadějná. Nabízejí se tyto možnosti, co si počít:

1. *Spokojit se s málem.* Nejsou vstupy, pro které problém potřebujeme řešit, dostatečně malé, abychom si mohli dovolit použít algoritmus s exponenciální složitostí? Zvlášť když takový algoritmus vylepšíme prořezáváním neperspektivních větví výpočtu a třeba ho i paralelizujeme.
2. *Vyřešit speciální případ.* Nemají naše vstupy nějaký speciální tvar, kterého bychom mohli využít? Grafové problémy jsou často v **P** třeba pro stromy nebo i obecněji pro bipartitní grafy. U číselných problémů zase někdy pomůže, jsou-li čísla na vstupu dostatečně malá.
3. *Řešení aproximovat.* Opravdu potřebujeme optimální řešení? Nestačilo by nám o kousíček horší? Často existuje polynomiální algoritmus, který nalezne nejhůře c-krát horší řešení než je optimum, kde c je konstanta.
4. *Použít heuristiku.* Neumíme-li nic lepšího, můžeme sáhnout po některé z mnoha heuristických technik, které sice nic nezaručují, ale obvykle nějaké uspokojivé řešení najdou. Může pomoci třeba hladový algoritmus nebo genetické algoritmy. Často platí, že čím déle heuristiku necháme běžet, tím lepší řešení najde.
5. *Kombinace přístupů.* Mnohdy lze předchozí přístupy kombinovat: například použít aproximační algoritmus a poté jeho výsledek ještě heuristicky vylepšovat. Tak získáme řešení, které od optima zaručeně není moc daleko, a pokud budeme mít štěstí, bude se od něj lišit jen velmi málo.

Nyní si některé z těchto technik předvedeme na konkrétních příkladech.

Největší nezávislá množina ve stromu

Ukážeme, že hledání největší nezávislé množiny je snadné, pokud graf je strom.

Lemma: Buď T zakořeněný strom a ℓ jeho libovolný list. Pak alespoň jedna z největších nezávislých množin obsahuje ℓ.
Důkaz: Mějme největší nezávislou množinu M, která list ℓ neobsahuje. Podívejme se na otce p listu ℓ (kdyby neexistoval, je celý strom jednovrcholový a tvrzení triviální). Leží p v M? Pokud ne, mohli bychom do M přidat list ℓ a dostali bychom větší nezávislou množinu. V opačném případě z M odebereme otce p a nahradíme ho listem ℓ, čímž dostaneme stejně velkou nezávislou množinu obsahující ℓ. ♡

Algoritmus bude přímočaře používat toto lemma. Dostane na vstupu strom, ten zakoření a zvolí libovolný list. Tento list umístí do nezávislé množiny a jeho otce odebere, protože se nemůže v nezávislé množině vyskytovat. Toto bude opakovat, dokud nějaké vrcholy zbývají. (Graf se v průběhu může rozpadnout na více komponent, ale to nevadí.)

Tento algoritmus jistě pracuje v polynomiálním čase. Šikovnou implementací můžeme složitost snížit až na lineární. Například tak, že budeme udržovat seznam listů. My si ukážeme jinou lineární implementaci založenou na prohledávání do hloubky. Bude pracovat s polem značek M, v němž na počátku bude všude *false* a postupně obdrží *true* všechny prvky hledané nezávislé množiny.

Algoritmus NzMnaVeStromu
Vstup: Strom T s kořenem v, pole značek M.

1. $M[v] \leftarrow$ *true*.
2. Pokud je v list, skončíme.
3. Pro všechny syny w vrcholu v:
4. Zavoláme se rekurzivně na podstrom s kořenem w.
5. Pokud $M[w] =$ *true*, položíme $M[v] \leftarrow$ *false*.

Barvení intervalového grafu

Mějme n přednášek s určenými časy začátku a konce. Chceme je rozvrhnout do co nejmenšího počtu poslucháren tak, aby nikdy neprobíhaly dvě přednášky naráz v jedné místnosti.

Chceme tedy obarvit co nejmenším počtem barev graf, jehož vrcholy jsou časové intervaly a dvojice intervalů je spojena hranou, pokud má neprázdný průnik. Takovým grafům se říká *intervalové* a pro jejich barvení existuje pěkný polynomiální algoritmus.

Podobně jako jsme geometrické problémy řešili zametáním roviny, zde budeme „zametat přímku bodem“, tedy procházet ji zleva doprava, a všímat si událostí, což budou začátky a konce intervalů. Pro jednoduchost předpokládejme, že všechny souřadnice začátků a konců jsou navzájem různé.

Kdykoliv interval začne, přidělíme mu barvu. Až skončí, o barvě si poznamenáme, že je momentálně volná, a dalším intervalům budeme přednostně přidělovat volné barvy. Řečeno v pseudokódu:

Algoritmus BarveníIntervalů
Vstup: Intervaly $[x_1, y_1], \ldots, [x_n, y_n]$.

1. $b \leftarrow 0$ *(počet zatím použitých barev)*

2. $B \leftarrow \emptyset$ *(které barvy jsou momentálně volné)*
3. Setřídíme množinu všech x_i a y_i.
4. Procházíme všechna x_i a y_i ve vzestupném pořadí:
5. Narazíme-li na x_i:
6. Je-li $B \neq \emptyset$, odebereme jednu barvu z B a uložíme ji do c_i.
7. Jinak $b \leftarrow b + 1$ a $c_i \leftarrow b$.
8. Narazíme-li na y_i:
9. Vrátíme barvu c_i do B.

Výstup: Obarvení $c_1, \ldots, c_n$.

Analýza: Tento algoritmus má časovou složitost $\mathcal{O}(n \log n)$ kvůli třídění souřadnic. Samotné obarvování je lineární.

Ještě ovšem potřebujeme dokázat, že jsme použili minimální možný počet barev. Uvažujme okamžik, kdy proměnná b naposledy vzrostla. Tehdy začal interval a množina B byla prázdná, což znamená, že jsme $b - 1$ předchozích barev museli přidělit intervalům, jež začaly a dosud neskončily. Existuje tedy b různých intervalů, které mají společný bod (v grafu tvoří kliku), takže každé obarvení potřebuje alespoň b barev.

Problém batohu s malými čísly

Připomeňme si *problém batohu.* Jeho optimalizační verze vypadá takto: Je dána množina n předmětů s hmotnostmi $h_1, \ldots, h_n$ a cenami $c_1, \ldots, c_n$ a nosnost batohu H. Hledáme podmnožinu předmětů $P \subseteq \{1, \ldots, n\}$, která se vejde do batohu (tedy $h(P) = \sum_{i \in P} h_i \leq H$) a její cena $c(P) = \sum_{i \in P} c_i$ je největší možná.

Ukážeme algoritmus, jehož časová složitost bude polynomiální v počtu předmětů n a součtu všech cen $C = \sum_i c_i$.

Použijeme dynamické programování. Představme si problém omezený na prvních k předmětů. Označme $A_k(c)$ (kde $0 \leq c \leq C$) minimum z hmotností těch podmnožin jejichž cena je právě c; pokud žádná taková podmnožina neexistuje, položíme $A_k(c) = \infty$.

Tato A_k spočteme indukcí podle k: Pro $k = 0$ je určitě $A_0(0) = 0$ a $A_0(1) = \ldots = A_0(C) = \infty$. Pokud již známe A_{k-1}, spočítáme A_k následovně: $A_k(c)$ odpovídá nějaké podmnožině předmětů z $1, \ldots, k$. V této podmnožině jsme buďto k-tý předmět nepoužili, a pak je $A_k(c) = A_{k-1}(c)$, nebo použili, a tehdy bude $A_k(c) = A_{k-1}(c - c_k) + h_k$ (to samozřejmě jen pokud $c \geq c_k$). Z těchto dvou možností si vybereme tu, která dává množinu s menší hmotností:

$$A_k(c) = \min(A_{k-1}(c), A_{k-1}(c - c_k) + h_k).$$

Přechod od A_{k-1} k A_k tedy trvá $\mathcal{O}(C)$, od A_1 až k A_n se dopočítáme v čase $\mathcal{O}(Cn)$.

Jakmile získáme A_n, známe pro každou cenu příslušnou nejlehčí podmnožinu. Maximální cena množiny, která se vejde do batohu, je tedy největší c^*, pro něž je $A_n(c^*) \leq H$. Jeho nalezení nás stojí čas $\mathcal{O}(C)$.

Zbývá zjistit, které předměty do nalezené množiny patří. Upravíme algoritmus, aby si pro každé $A_k(c)$ pamatoval ještě $B_k(c)$, což bude index posledního předmětu,

který jsme do příslušné množiny přidali. Pro nalezené c^* tedy bude $i = B_n(c^*)$ poslední předmět v nalezené množině, $i' = B_{i-1}(c^* - c_i)$ ten předposlední a tak dále. Takto v čase $\mathcal{O}(n)$ rekonstruujeme celou množinu od posledního prvku k prvnímu.

Máme tedy algoritmus, který vyřeší problém batohu v čase $\mathcal{O}(nC)$. Tato funkce ovšem není polynomem ve velikosti vstupu, jelikož reprezentujeme-li vstup binárně, C může být až exponenciálně velké vzhledem k délce jeho zápisu. To je pěkný příklad tzv. *pseudopolynomiálního* algoritmu, tedy algoritmu, jehož složitost je polynomem v počtu čísel na vstupu a jejich velikosti. Pro některé **NP**-úplné problémy takové algoritmy existují, pro jiné (např. pro nezávislou množinu) by z jejich existence plynulo **P** = **NP**.

Verze bez cen: Jednodušší verzi problému batohu, která nerozlišuje mezi hmotnostmi a cenami, zvládneme i jiným algoritmem, opět založeným na dynamickém programování.

Indukcí podle k vytváříme množiny Z_k obsahující všechny hmotnosti menší než H, kterých nabývá nějaká podmnožina prvních k prvků. Jistě je $Z_0 = \{0\}$. Podobnou úvahou jako v předchozím algoritmu dostaneme, že každou další Z_k můžeme zapsat jako sjednocení Z_{k-1} s kopií Z_{k-1} posunutou o h_k, ignorujíce hodnoty větší než H. Nakonec ze Z_n vyčteme výsledek.

Všechny množiny přitom mají nejvýše $H + 1$ prvků, takže pokud si je budeme udržovat jako setříděné seznamy, spočítáme sjednocení sléváním v čase $\mathcal{O}(H)$ a celý algoritmus doběhne v čase $\mathcal{O}(Hn)$.

Aproximace problému obchodního cestujícího

V *problému obchodního cestujícího* je zadán neorientovaný graf G, jehož hrany jsou ohodnoceny délkami $\ell(e) \geq 0$. V tomto grafu chceme nalézt nejkratší z hamiltonovských kružnic, tedy těch, které navštíví všechny vrcholy.

Není překvapivé, že tento problém je těžký – už sama existence hamiltonovské kružnice je **NP**-úplná. Ukážeme ovšem, že pokud je graf úplný a platí v něm trojúhelníková nerovnost (tj. $\ell(x,z) \leq \ell(x,y) + \ell(y,z)$ pro všechny trojice vrcholů x, y, z), můžeme problém obchodního cestujícího 2-aproximovat, tedy najít v polynomiálním čase kružnici, která je přinejhorším dvakrát delší než ta optimální.

Grafy s trojúhelníkovou nerovností přitom nejsou nijak neobvyklé – odpovídají totiž konečným metrickým prostorům.

Algoritmus bude snadný: Najdeme nejmenší kostru a obchodnímu cestujícímu poradíme, ať ji obejde. To můžeme popsat například tak, že kostru zakořeníme, prohledáme ji do hloubky a zaznamenáme, jak jsme procházeli hranami. Každou hranou kostry přitom projdeme dvakrát – jednou dolů, podruhé nahoru. Tím však nedostaneme kružnici, nýbrž jen nějaký uzavřený sled, protože vrcholy navštěvujeme vícekrát. Sled tedy upravíme tak, že kdykoliv se dostává do již navštíveného vrcholu, přeskočí ho a přesune se až do nejbližšího dalšího nenavštíveného. Tím ze sledu vytvoříme hamiltonovskou kružnici a jelikož v grafu platí trojúhelníková nerovnost, celková délka nevzrostla. (Pořadí vrcholů na kružnici můžeme získat také tak, že během prohledávání budeme vypisovat vrcholy v preorderu. Rozmyslete si, že je totéž.)

Věta: Nalezená kružnice není delší než dvojnásobek optima.
Důkaz: Označme T délku minimální kostry, A délku kružnice vydané naším algoritmem a O (optimum) délku nejkratší hamiltonovské kružnice. Z toho, jak jsme kružnici vytvořili, víme, že $A \leq 2T$. Platí ovšem také $T \leq O$, jelikož z každé hamiltonovské kružnice vznikne vynecháním hrany kostra a ta nemůže být menší než minimální kostra. Složením obou nerovností získáme $A \leq 2T \leq 2O$. ♡

Sestrojili jsme tedy 2-aproximační algoritmus pro problém obchodního cestujícího. Dodejme ještě, že trochu složitějším trikem lze tento problém 1.5-aproximovat a že v některých metrických prostorech (třeba v euklidovské rovině) lze v polynomiálním čase najít $(1+\varepsilon)$-aproximaci pro libovolné $\varepsilon > 0$. Ovšem čím menší ε, tím déle algoritmus poběží.

Trojúhelníková nerovnost nicméně byla pro tento algoritmus naprosto nezbytná. To není náhoda – hned dokážeme, že bez tohoto předpokladu je libovolná aproximace stejně těžká jako přesné řešení.

Věta: Pokud pro nějaké reálné $t \geq 1$ existuje polynomiální t-aproximační algoritmus pro problém obchodního cestujícího bez trojúhelníkové nerovnosti, pak je **P** = **NP**.
Důkaz: Ukážeme, že pomocí takového aproximačního algoritmu dokážeme v polynomiálním čase zjistit, zda v grafu existuje hamiltonovská kružnice, což je **NP**-úplný problém.

Dostali jsme graf G, ve kterém hledáme hamiltonovskou kružnici (zkráceně HK). Doplníme G na úplný graf G'. Všem původním hranám nastavíme délku na 1, těm novým na nějaké dost velké číslo c. Kolik to bude, určíme za chvíli.

Graf G' je úplný, takže v něm určitě nějaké HK existují. Ty, které se vyskytují i v původním grafu G, mají délku přesně n. Jakmile ale použijeme jedinou hranu, která z G nepochází, vzroste délka kružnice alespoň na $n - 1 + c$.

Podle délky nejkratší HK tedy dokážeme rozpoznat, zda existuje HK v G. Potřebujeme ovšem zjistit i přes zkreslení způsobené aproximací. Musí tedy platit $tn < n - 1 + c$. To snadno zajistíme volbou hodnoty c větší než $(t-1)n + 1$.

Naše konstrukce přidala polynomiálně mnoho hran s polynomiálně velkým ohodnocením, takže graf G' je polynomiálně velký vzhledem ke G. Rozhodujeme tedy existenci HK v polynomiálním čase a **P** = **NP**. ♡

Poznámka: Podobně můžeme dokázat, že pokud **P** ≠ **NP**, neexistuje pro problém obchodního cestujícího ani pseudopolynomiální algoritmus. Stačí původním hranám přiřadit délku 1 a novým délku 2.

Aproximační schéma pro problém batohu

Již víme, jak optimalizační verzi problému batohu vyřešit v čase $\mathcal{O}(nC)$, pokud jsou hmotnosti i ceny na vstupu přirozená čísla a C je součet všech cen. Jak si poradit, pokud je C obrovské? Kdybychom měli štěstí a všechny ceny byly násobky nějakého čísla p, mohli bychom je tímto číslem vydělit. Tak bychom dostali zadání s menšími čísly, jehož řešením by byla stejná množina předmětů jako u zadání původního.

Když nám štěstí přát nebude, můžeme přesto zkusit ceny vydělit a výsledky nějak zaokrouhlit. Optimální řešení nové úlohy pak sice nemusí odpovídat optimál-

nímu řešení té původní, ale když nastavíme parametry správně, bude alespoň jeho dobrou aproximací.

Základní myšlenka: Označíme $c_{\max}$ maximum z cen c_i. Zvolíme nějaké přirozené číslo $M < c_{\max}$ a zobrazíme interval cen $[0, c_{\max}]$ na $\{0, \ldots, M\}$ (tedy každou cenu znásobíme poměrem $M/c_{\max}$ a zaokrouhlíme). Jak jsme tím zkreslili výsledek? Všimněme si, že efekt je stejný, jako kdybychom jednotlivé ceny zaokrouhlili na násobky čísla $c_{\max}/M$ (prvky z intervalu $[i \cdot c_{\max}/M, (i+1) \cdot c_{\max}/M)$ se zobrazí na stejný prvek). Každé c_i jsme tím tedy změnili o nejvýše $c_{\max}/M$, celkovou cenu libovolné podmnožiny předmětů pak nejvýše o $n \cdot c_{\max}/M$. Navíc odstraníme-li ze vstupu předměty, které se samy nevejdou do batohu, má optimální řešení původní úlohy cenu $c^* \geq c_{\max}$, takže chyba naší aproximace nepřesáhne $n \cdot c^*/M$. Má-li tato chyba být shora omezena $\varepsilon \cdot c^*$, musíme zvolit $M \geq n/\varepsilon$.

Na této myšlence „kvantování cen“ je založen následující algoritmus.

Algoritmus AproximaceBatohu

1. Odstraníme ze vstupu všechny předměty těžší než H.
2. Spočítáme $c_{\max} = \max_i c_i$ a zvolíme $M = \lceil n/\varepsilon \rceil$.
3. Kvantujeme ceny: Pro $i = 1, \ldots, n$ položíme $\hat{c}_i \leftarrow \lfloor c_i \cdot M/c_{\max} \rfloor$.
4. Vyřešíme dynamickým programováním problém batohu pro upravené ceny $\hat{c}_1, \ldots, \hat{c}_n$ a původní hmotnosti i kapacitu batohu.
5. Vybereme stejné předměty, jaké použilo optimální řešení kvantovaného zadání.

Analýza: Kroky 1–3 a 5 jistě zvládneme v čase $\mathcal{O}(n)$. Krok 4 řeší problém batohu se součtem cen $\hat{C} \leq nM = \mathcal{O}(n^2/\varepsilon)$, což stihne v čase $\mathcal{O}(n\hat{C}) = \mathcal{O}(n^3/\varepsilon)$. Zbývá dokázat, že výsledek našeho algoritmu má opravdu relativní chybu nejvýše ε.

Označme P množinu předmětů použitých v optimálním řešení původní úlohy a $c(P)$ cenu tohoto řešení. Podobně Q bude množina předmětů v optimálním řešení nakvantované úlohy a $\hat{c}(Q)$ jeho hodnota v nakvantovaných cenách. Potřebujeme odhadnout ohodnocení množiny Q v původních cenách, tedy $c(Q)$, a srovnat ho s $c(P)$.

Nejprve ukážeme, jakou cenu má optimální řešení P původní úlohy v nakvantovaných cenách:

$$\hat{c}(P) = \sum_{i \in P} \hat{c}_i = \sum_{i \in P} \left\lfloor c_i \cdot \frac{M}{c_{\max}} \right\rfloor \geq \sum_{i \in P} \left(c_i \cdot \frac{M}{c_{\max}} - 1 \right) \geq$$
$$\geq \left(\sum_{i \in P} c_i \cdot \frac{M}{c_{\max}} \right) - n = c(P) \cdot \frac{M}{c_{\max}} - n.$$

Nyní naopak spočítejme, jak dopadne optimální řešení Q nakvantovaného problému při přepočtu na původní ceny (to je výsledek našeho algoritmu):

$$c(Q) = \sum_{i \in Q} c_i \geq \sum_{i \in Q} \hat{c}_i \cdot \frac{c_{\max}}{M} = \left(\sum_i \hat{c}_i \right) \cdot \frac{c_{\max}}{M} = \hat{c}(Q) \cdot \frac{c_{\max}}{M} \geq \hat{c}(P) \cdot \frac{c_{\max}}{M}.$$

Poslední nerovnost platí proto, že $\hat{c}(Q)$ je optimální řešení kvantované úlohy, zatímco $\hat{c}(P)$ je nějaké další řešení téže úlohy, které nemůže být lepší.[1] Teď už stačí složit obě nerovnosti a dosadit za M:

$$c(Q) \geq \left(\frac{c(P)\cdot M}{c_{\max}} - n\right)\cdot\frac{c_{\max}}{M} \geq c(P) - \frac{n\cdot c_{\max}}{n/\varepsilon} \geq c(P) - \varepsilon c_{\max} \geq$$
$$\geq c(P) - \varepsilon c(P) = (1-\varepsilon)\cdot c(P).$$

Na přechodu mezi řádky jsme využili toho, že každý předmět se vejde do batohu, takže optimum musí být alespoň tak cenné jako nejcennější z předmětů.

Shrňme, co jsme dokázali:

Věta: Existuje algoritmus, který pro každé $\varepsilon > 0$ nalezne $(1-\varepsilon)$-*aproximaci* problému batohu s n předměty v čase $\mathcal{O}(n^3/\varepsilon)$.

Dodejme ještě, že algoritmům, které dovedou pro každé $\varepsilon > 0$ najít v polynomiálním čase $(1-\varepsilon)$-aproximaci optimálního řešení, říkáme *polynomiální aproximační schémata* (PTAS – Polynomial-Time Approximation Scheme). V našem případě je dokonce složitost polynomiální i v závislosti na $1/\varepsilon$, takže schéma je *plně polynomiální* (FPTAS – Fully Polynomial-Time Approximation Scheme).

Cvičení

1. Popište polynomiální algoritmus pro hledání nejmenšího vrcholového pokrytí stromu. (To je množina vrcholů, která obsahuje alespoň jeden vrchol z každé hrany.)
2.* Nalezněte polynomiální algoritmus pro hledání nejmenšího vrcholového pokrytí bipartitního grafu.
3. Ukažte, jak v polynomiálně najít největší nezávislou množinu v intervalovém grafu.
4.* Vyřešte v polynomiálním čase 2-SAT, tedy splnitelnost formulí zadaných v CNF, jejichž klauzule obsahují nejvýše 2 literály.
5. Problém E3,E3-SAT je zesílením 3,3-SATu. Chceme zjistit splnitelnost formule v CNF, jejíž každá klauzule obsahuje právě tři různé proměnné a každá proměnná se nachází v právě třech klauzulích. Ukažte, že tento problém lze řešit efektivně z toho prostého důvodu, že každá taková formule je splnitelná.
6. Pokusíme se řešit problém dvou loupežníků hladovým algoritmem. Probíráme předměty od nejdražšího k nejlevnějšímu a každý dáme tomu loupežníkovi, který má zrovna méně. Je nalezené řešení optimální?
7. Problém tří loupežníků: Je dána množina předmětů s cenami, chceme ji rozdělit na 3 části o stejné ceně. Navrhněte pseudopolynomiální algoritmus.
8. Problém MaxCut: vrcholy zadaného grafu chceme rozdělit do dvou množin tak, aby mezi množinami vedlo co nejvíce hran. Jinými slovy chceme nalézt bipartitní

[1] Zde nás zachraňuje, že ačkoliv u obou úloh leží optimum obecně jinde, obě mají stejnou množinu *přípustných řešení,* tedy těch, která se vejdou do batohu. Kdybychom místo cen kvantovali hmotnosti, nebyla by to pravda a algoritmus by nefungoval.

podgraf s co nejvíce hranami. Rozhodovací verze tohoto problému je **NP**-úplná, zkuste jej v polynomiálním čase 2-aproximovat.

9.* V problému MaxE3-SAT dostaneme formuli v CNF, jejíž každá klauzule obsahuje právě 3 různé proměnné, a chceme nalézt ohodnocení proměnných, při němž je splněno co nejvíce klauzulí. Rozhodovací verze je **NP**-úplná. Ukažte, že při náhodném ohodnocení proměnných je splněno v průměru 7/8 klauzulí. Z toho odvoďte deterministickou 7/8-aproximaci v polynomiálním čase.

10. Hledejme vrcholové pokrytí následujícím hladovým algoritmem. V každém kroku vybereme vrchol nejvyššího stupně, přidáme ho do pokrytí a odstraníme ho z grafu i se všemi již pokrytými hranami. Je nalezené pokrytí nejmenší? Nebo alespoň $\mathcal{O}(1)$-aproximace nejmenšího?

11.* Uvažujme následující algoritmus pro nejmenší vrcholové pokrytí grafu. Graf projdeme do hloubky, do výstupu vložíme všechny vrcholy vzniklého DFS stromu kromě listů. Dokažte, že vznikne vrcholové pokrytí a že 2-aproximuje to nejmenší.

12.* V daném orientovaném grafu hledáme acyklický podgraf s co nejvíce hranami. Navrhněte polynomiální 2-aproximační algoritmus.

Nápovědy ke cvičením

2. Vzpomeňte si na síť z algoritmu na největší párování. Jak v ní vypadají řezy?
4. Na každou klauzuli se můžeme podívat jako na implikaci.
5. Použijte Hallovu větu.
9. Linearita střední hodnoty.
11. Najděte v G párování obsahující alespoň tolik hran, kolik je polovina počtu vrcholů vráceného pokrytí. Jak velikost párování souvisí s velikostí nejmenšího vrcholového pokrytí?
12. Libovolné očíslování vrcholů rozdělí hrany na „dopředné“ a „zpětné“.