3. CVIČENÍ Z ÚVODU DO APROXIMACÍ – DODATEK řešení prvních dvou příkladů

PŘÍKLAD PRVNÍ Mějme klasický NP-těžký PROBLÉM BATOHU, to jest n objektů $a_1, \ldots, a_n$, kde každý objekt a_i má váhu w_i a cenu c_i, a my máme batoh o nosnosti B.

Navrhněte hladový 2-aproximační algoritmus pro tento problém. (Existují i o dost lepší nehladové aproximační algoritmy.)

Řešení. Nejprve nahlédneme, že hladový postup (jakmile objekt zabalíme, už ho nevyhodíme) nefunguje, pokud si objekty nepředtřídíme: stačí nám jeden objekt váhy 1 a ceny 0 a jeden váhy B a ceny 1.

Druhý pokus: Setřídíme si tedy objekty podle hustoty (c_i/w_i) sestupně a budeme objekty hladově brát, dokud se vejdou.

Bohužel, i druhý pokus selže: uvažme jeden objekt váhy 1 a ceny 2 a druhý objekt váhy B a ceny B. První objekt je hustší, ale pokud ho vezmeme, tak budeme nejvýše $B/2$-aproximačním algoritmem, což je nepříjemné, když B je součást vstupu a může být libovolně velké.

Dobrá, tak třetí pokus: postupujme hladově podle hustoty, a když narazíme na první problémový objekt P (první podle hustoty, který se hladovému algoritmu nevejde do batohu) tak vybereme lepší řešení – buď vezmeme do batohu P samotné, nebo P ignorujeme a necháme si dosavadní objekty (říkejme jim $\mathcal{A}$) – podle toho, která možnost má větší cenu.

Teď bychom mohli s algoritmem pokračovat dál, ale pojďme místo toho začít analýzu a zjistit, jak dobrý algoritmus je ihned po přijetí/odmítnutí P.

Analýzu začneme rozborem případů podle objektu P. Nevíme, jak optimální rozložení batohu (po zpracování celého vstupu) vypadá, ale o optimálním rozložení OPT můžeme předpokládat následující:

1. V OPT jsou objekty setříděny také podle hustoty.
2. Objekt P v OPT buď je, nebo není.

Když P v OPT je, rozdělíme objekty v OPT do dvou hromádek: v jedné hromádce je P, v druhé je $\mathcal{R} = OPT \setminus P$. Jak dobrý je náš hladový algoritmus (který jsme zatím zastavili po přijetí P) oproti cenám těchto hromádek?

Vůči první hromádce P je samozřejmě dobře: $\max(c_P, c_\mathcal{A}) \geq c_P$. V porovnání s druhou hromádkou $\mathcal{R}$ použijeme její srovnání s objekty v $\mathcal{A}$. Hromádka $\mathcal{R}$ je v OPT spolu s P, ale hromádka $\mathcal{A}$ se s P nevešla – takže musí platit, že buď $\mathcal{R}$ obsahuje řidší objekty, nebo v ní nějaké objekty z $\mathcal{A}$ chybí. Jinými slovy musí platit, že $\mathcal{R}$ je menší ceny (nebo rovné) $\mathcal{A}$, a tedy $\max(c_P, c_\mathcal{A}) \geq c_\mathcal{R}$. Tím jsme dostali, že $c_{OPT} = c_P + c_\mathcal{R} \leq 2\max(c_P, c_\mathcal{A})$, čili v tomto případě už máme 2-aproximační algoritmus, i když se po zpracování P zastavíme.

Když P v OPT není, rozdělíme objekty v OPT do jiných dvou hromádek: hromádku $\mathcal{S}$ definujeme jako „objekty v OPT setříděné podle hustoty do celkové váhy přesně $w_\mathcal{A}$“, a $\mathcal{T}$ jako zbytek. Zde uvažujeme, že do $\mathcal{S}$ vložíme i zlomek objektu, který doplní váhu tak, aby $w_\mathcal{S} = w_\mathcal{A}$, jako kdybychom batoh řízli pilou, a zbytek tohoto objektu vložíme do $\mathcal{T}$.

Analýza bude postupovat obdobně – zase odhadneme obě hromádky zvlášť. První hromádka $\mathcal{S}$ se dá snadno odhadnout $\mathcal{A}$ – obě jsou stejné váhy a do $\mathcal{A}$ jsme hladově vkládali ty nejhustší objekty, takže v $\mathcal{S}$ jsou buď stejné, nebo řidší objekty, ale tak nebo onak platí $\max(c_P, c_\mathcal{A}) \geq c_\mathcal{S}$.

Druhá hromádka se (opět) vztáhne k P – protože v OPT objekt P není, budou v hromádce $\mathcal{T}$ objekty jen řidší než P (nezapomeňme, P byl nejhustší objekt, který se nevešel k $\mathcal{A}$). No a váha $w_\mathcal{T} \leq w_P$, protože P se nevešel společně do batohu s váhou $w_\mathcal{A}$, a tedy se nevejde ani se stejnou váhou $w_\mathcal{S}$ (což je $B - w_\mathcal{T}$).

Zase tedy máme $\max(c_P, c_\mathcal{A}) \geq c_\mathcal{T}$ a i v tomto případě už máme 2-aproximační algoritmus. Dozvěděli jsme se, že opravdu stačí tento algoritmus:

Postupujme hladově podle hustoty, a když narazíme na první problémový objekt P (první podle hustoty, který se hladovému algoritmu nevejde do batohu) tak vybereme lepší řešení – buď vezmeme do batohu P samotné, nebo P ignorujeme a necháme si dosavadní objekty (říkejme jim $\mathcal{A}$) – podle toho, která možnost má větší cenu. Jakmile P zpracujeme, už se dalšími objekty nezabýváme.

PŘÍKLAD DRUHÝ Uvažujme ROZVRHOVÁNÍ SE ZÁVISLOSTMI: rozvrhujeme úlohy různých délek na m počítačů (m součástí vstupu), ale navíc máme závislosti mezi úlohami. Přesněji, na úlohách je definovaný acyklický graf a úlohu je možné zahájit až ve chvíli, kdy jsou všechny úlohy jí předcházející v grafu dokončeny. Navrhněte hladový algoritmus a dokažte, že je 2-aproximační.

Řešení. Algoritmus bude podobný jako v případě rozrhování bez závislostí:

- Budeme si průběžně udržovat množinu rozvrhnutelných úloh, které už mají závislosti splněny (například odmazáváním vrcholů z grafu závislostí, a sledování vrcholů se vstupním stupněm 0).
- Jakmile je dokončen libovolný úkol na libovolném počítači, ihned spustíme libovolný nový rozvrhnutelný úkol. Jinak řečeno, nerozvrhneme nic jen tehdy, pokud všechny zbývající úkoly jsou nerozvrhnutelné.

V analýze hladového algoritmu pro rozvrhování jsme využili dva dolní odhady:

1. $OPT \geq$ celkový objem úloh $/m$.
2. $OPT \geq p_j$ pro libovolnou úlohu j.

Tyto odhady platí i v náší úloze. Objevil se ale nějaký nový? Ano:

3. $OPT \geq$ délka libovolného řetězu úloh, kde řetěz chápeme jako orientovanou cestu v grafu závislostí. Jeho délka je pak součet délek úloh na řetězu.

Ještě je dobré si uvědomit, že bez újmy na obecnosti můžeme předpokládat, že graf závislostí je *tranzitivně zúplněn* – pokud úloha A závisí na B, B závisí na C, tak rovnou přidáme i závislostní hranu z A do C.

Nyní analyzujme náš algoritmus. V analýze obecného rozvrhování jsme rozdělili délku výpočtu na dva díly, kde každý šel odhadnout pomocí dolního odhadu na OPT – jeden díl byl *délka výpočtu úlohy, která doběhne poslední* a druhý díl byl *doba, kdy stroje jsou všechny plně zatíženy*. V analýze jsme pak jen zdůvodnili, proč toto je opravdu rozložení celého běhu.

V našem případě se závislostmi skoro jistě můžeme použít díl druhý: *doba, kdy stroje jsou všechny plně zatíženy*. Nicméně zbytek běhu už nemusí být jen na konci – merezy v rozvrhu se mohou objevovat i během výpočtu. Odhadneme je tak, že sestavíme vhodný řetěz, který je všechny pokrývá.

Řetěz začneme poslední úlohou, která jistě může běžet i podél nějaké mezery. Pokračovat budeme tak, že najdeme poslední mezeru (čili mezeru nejbližší poslední úloze), která se stala před spuštěním samotné poslední úlohy.

Sokraticky se zeptáme: „Proč neběžela poslední úloha v této mezeře?“ Musí to být proto, neboť je poslední uloha p_l závislá na nějaké úloze p_k, která zrovna běží v celé délce této mezery. Do řetězu tedy přidáme p_k a postupujeme induktivně: najdeme poslední mezeru před tím, než p_k běží, a pokračujeme stejným způsobem.

Rozdělení na *úlohy v našem induktivním řetězu* a na *dobu, kdy stroje jsou plně zatíženy* už pokrývá celou dobu běhu, a protože každý můžeme odhadnout zezdola OPT, dokázali jsme 2-aproximativnost našeho algoritmu.