1. CVIČENÍ Z APROXIMACÍ

pí-tasy

V prvních několika příkladech se budeme zabývat problémem batohu. Začněme zlehka:

PŘÍKLAD PRVNÍ Mějme PROBLÉM BATOHU Z PROGRAMOVÁNÍ 1 – na vstupu máme objekty i s celočíselnou vahou w_i, limit B, a ptáme se, kolik jich nejvýše můžeme vložit do batohu, abychom nepřesáhli limit. Připomeňte si pseudopolynomiální algoritmus.

Ve většině případů se ale setkáme s PROBLÉMEM BATOHU formulovaným takto: máme na vstupu n objektů s celočíselnou vahou w_i a cenou c_i, a limit B, a chceme najít objekty co největší ceny, které se do batohu ještě vejdou.

PŘÍKLAD DRUHÝ Navrhněte algoritmus pro problém batohu, který je pseudopolynomiální v obou parametrech, tj. má složitost $O(n \cdot \min(W, C))$, kde $C = \sum_{j\in[n]} c_j$ a $W = \sum_{j\in[n]} w_j$.

PŘÍKLAD TŘETÍ

Asi víte, kam směřujeme – nyní chceme vymyslet FPTAS pro problém batohu (možná jste jej už slyšeli na Složitosti). Dost možná víte, že se bude zaokrouhlovat jedna z proměnných – konkrétně budeme mít nějaký škálovací faktor μ a budeme chtít nastavit *škálované ceny* $c'_i = \lfloor \frac{c_i}{\mu} \rfloor$. Potom, co ceny naškálujeme, vyřešíme problém dynamickým programováním a pak se budeme koukat na *skoro-původní ceny* $\tilde{c}_i = \mu \cdot c'_i$.

Rozdělme si nyní práci do čtyř kroků:

1. Odhadněte $\tilde{c}_i$ zezdola i zeshora pomocí původních cen c_i (a μ).
2. Navrhněte (ne)rovnici, která bude definovat škálovací faktor μ pomocí OPT a ε (náš parametr ze začátku). Rovnici navrhněte tak, aby když by platila, tak výsledné řešení dynamického programu (naškálované zpátky) bude velikosti aspoň $(1 - \varepsilon)OPT$.
3. Nevýhodou nerovnice z předchozího bodu je to, že je závislá na OPT, které dopředu neznáme. Použijeme tedy dolní odhad $OPT \geq c_{\max}$ a tím máme plně definované μ. Rozmyslete si, proč chceme dolní odhad na OPT.
4. Nakonec spočítejte, jakou bude mít složitost dynamické programování, které zavoláme na naškálované hodnoty.

Odbočka k hladovým algoritmům na PROBLÉM BATOHU:

PŘÍKLAD ČTVRTÝ

1. Rozmyslete si, proč „naivně hladový algoritmus“, to jest „vložím nejdražší objekt, co se vejde, do batohu, a pokračuji dále“, je špatným.
2. OK, tak zkusme tento algoritmus: setřiďme si objekty podle „hustoty“ (poměr cena/velikost), procházejme je od nejhustšího a berme jen ty, které se vejdou. Prozradím vám, že i tento algoritmus nebude mít dobrý výsledek. Najděte vstup, kde selže.
3. Tak nakonec: setřídíme si objekty podle hustoty, přidávejme je, dokud to jde, a když najdeme objekt j, který již přidat nelze, tak si vybereme lepší z řešení $\{1, \ldots, j-1\}$ a $\{j\}$. Ukažte, že je to $\frac{1}{2}$-aproximace (nebo, chcete-li, 2-aproximace).

Ta odbočka nebyla samoúčelná – můžeme totiž použít 2-aproximační algoritmus místo poměrně špatného dolního odhadu na optimum, které jsme měli předtím.

PŘÍKLAD PÁTÝ Upravte škálování a analýzu PTASu pro PROBLÉM BATOHU tak, abyste použili lepší odhad na optimum a ušetřili jeden faktor n v analýze časové složitosti PTASu.

Nápověda: Podívejte se zpátky na druhý příklad, kde jsme dělali pseudopolynomiální algoritmus

pro batoh. Vypozorujte a dokažte, že ve skutečnosti běží v čase $O(n \cdot \min(C^*, B))$, kde B je limit batohu a C^* cena optimálního řešení problému.

PŘÍKLAD ŠESTÝ Mějme VÁŽENÉ ROZVRHOVÁNÍ NA JEDNOM STROJI S DEADLINY – úkoly mají délku p_j, váhu w_j a termín d_j, a my chceme maximalizovat vážený součet úkolů, které na jednom počítači doběhnou do svého termínu.

Nejprve vymyslete dynamický program, který běží v čase $O(nW)$, kde W je součet vah...

PŘÍKLAD SEDMÝ ... a pak, nepřekvapivě, vymyslete FPTAS pro tento problém.

PŘÍKLAD OSMÝ V problému NEJLEVNĚJŠÍ CESTA DO LIMITU máme zadaný acyklický orientovaný graf G s hranami, které mají celočíselnou cenu c_{ij} a dobu přesunu t_{ij} – jak dlouho trvá přesunout se z i do j. Máme také limit T, zadaný zdroj s a cíl t a naším úkolem je najít nejlevnější cestu z s do t, která bude trvat celkově pod limit T.

Navrhněte FPTAS pro tento problém.

Nápověda: Opět se hodí rozmyslet si dynamické programy pro tento problém. Jedno z možných řešení je vytvořit testovací algoritmus, který buď najde cestu ceny C nebo zahlásí, že nejkratší je ceny alespoň $(1 + \varepsilon)C$ – a pak binárně vyhledávat jako u rozvrhování na paralelních strojích.