4 7. dubna – Zvyšování efektivity rekurze, memoizace, dynamické programování

Na slidu máte vyjmenováno plno problémů, z nichž některé jste se v zimním semestru učili řešit s využitím rekurze. Konkrétně a například výpočet Fibonacciho čísel, výpočet Catalanových čísel, výpočet kombinačních čísel. Nyní již by vám nemělo činit potíže konkrétně tyto úlohy vyřešit za využití rekurze.

Všimněte si však, že rekurze je velmi neefektivní. Všechna ta čísla počítá přičítáním jedničky (a jak každý ví – a ostatní nám to uvěří – všechna ta čísla umějí růst exponenciálně). My začneme tím, že si všimneme toho, že rekurze je parametrizována několika málo argumenty (které dotyčná funkce dostane) – tyto argumenty jsou obvykle jeden nebo dva a výsledek volání funkce závisí pouze na těchto parametrech (funkce si sama nic nenačítá).

Za těchto okolností můžeme udělat tuto úvahu: Máme-li znovu počítat $f(n)$, nebude jednodušší si tento údaj zapamatovat? Usoudíme, že ano. Proto si vytvoříme cache na výsledky tak, abychom pro všechny myslitelné parametry funkce měli místo v cachi. Datový typ bude odpovídat tomu typu, co funkce počítá (nejspíše 32bitovému integeru). Nyní pokaždé, když je dotyčná funkce zavolána, tato funkce napřed koukne do cache. Zjistí-li, že je výsledek výpočtu s těmito parametry znám, nic nepočítá a pouze vrátí hodnotu z cache. Vyzkoušejte si toto na Fibonacciho číslech. Všimněte si, že necháte-li si spočítat třeba 80. Fibonacciho číslo (k čemuž už budete potřebovat 64bitový integer, tedy long), výpočet bez cache nedoběhne, kdežto ten s cachí má výsledek téměř okamžitě.

Přesně totéž provedeme i s ostatními problémy. Tedy rekurzi vybavíme cachí. Tomuto říkáme memoizace. Konkrétní provedení pro jednotlivé problémy si prohlédněte ve slidech.

Nejdelší rostoucí podposloupnost též můžeme hledat s využitím rekurze (algoritmem popsaným ve slidech). Tím vyzkoušíme všechny podposloupnosti. Nicméně funkce hledající nejdelší rostoucí podposloupnost končící v i. prvku bude stále nacházet znovu tutéž posloupnost (potažmo její délku). Tak údaje nacachujeme.

Se závorkováním matic uděláme úplně totéž. Tomuto myšlenkovému schématu říkáme memoizace.

Povšimneme-li si, jakým způsobem se cache vyplňuje, zjistíme, že rekurzi vůbec nepotřebujeme a cache vyplníme správným směrem na základě hodnot v ní obsažených. Tomu již říkáme Dynamické programování (toto bude předneseno na Algoritmech a datových strukturách). Nakonec můžeme ještě cache zoptimalizovat (jako na posledním slidu). Toto myšlenkové schéma nám ukazuje, jak souvisí rekurzivní algoritmus výpočtu Fibonacciho čísel s tím algoritmem, který každý normální člověk použije (tedy s cyklem a třemi či čtyřmi proměnnými).

Memoizace u množství problémů reprezentuje rozdíl mezi algoritmem zcela nepoužitelným a algoritmem velmi slušným.

Cvičení:* Funkci (rekurzivní), která vyplňuje cache, modifikujte tak, aby při každém zavolání ukázala, jak nyní vypadá cache (tedy aby celou cache vypsala

na standardní výstup). Sledujte, z jaké strany se cache plní a jakým způsobem. Z toho uvidíte, jak bude vypadat algoritmus dynamického programování.