In-place Mergesort

Dejdar Jan

8. 11. 2016

1 Definice problému

Jednou z nevýhod populárního třídícího algoritmu Merge sort je paměťová složitost $O(n \log n)$ u neoptimalizované verze, případně $O(n)$ u implementace, která si pracovní pole alokuje dopředu. Existuje však i in-place verze tohoto algoritmu, kterou se zde budu zabývat.

Efektivitu algoritmu se pokusím zvýšit použitím algoritmu Insertion sort pro malá pole. Pokusím se najít optimální mez pro spuštění Insertion sortu. Dále budu zkoumat vliv rozložení vstupní posloupnosti. Změřím časy jednotlivých verzí algoritmu pro náhodnou a obráceně seřazenou vstupní posloupnost.

1.1 Popis sekvenčního algoritmu

Klasický Merge sort používá přídavné pomocné pole pro slévaní dvou seřazených částí pole. V implementaci in-place verze budeme muset použít zbytek původního pole jako pracovní prostor pro toto slévání. Prvky v tomto zbytku pole ale není možné přepsat, protože je budeme řadit později. Myšlenka jak toho docílit spočívá v tom, že když chceme menší z dvou právě porovnávaných prvků umístit do pracovního prostoru, vyměníme tento prvek s příslušným prvkem v pracovním prostoru. Po skončení slévání tedy dvě původně seřazená podpole obsahují prvky, které předtím byly v pracovním prostoru, zatímco tento obsahuje seřazené pole.

Při slévání musí být splněny dvě podmínky:

1. Pracovní prostor musí být dostatečně velký, aby pojal obě části pole, které chceme slévat.

2. Pracovní prostor se může překrývat se seřazenými podpoli, ale je nutné zajistit, že nepřepíšeme žádné neslité prvky.

S výše uvedeným postupem je triviální sestrojit algoritmus, který seřadí polovinu původního pole, protože máme dostatek pracovního prostoru pro slévání seřazených prvků. Výsledek je znázorněn v tabulce 1. Otázkou zůstává, jak seřadit zbývající neseřazenou polovinu pole.

| ... SEŘAZENÉ ... | ... NESEŘAZENÉ ... |

Tabulka 1: Seřazená polovina pole

Intuitivní postup by byl rekurzivně seřadit druhou polovinu neseřazeného pole (viz tabulka 2), takže by zbyla jen $\frac{1}{4}$ neseřazeného pole. Problém je v tom, že musíme slít B a A, na což nemáme dostatečně velký pracovní prostor.

| ... NESEŘAZENÁ $\frac{1}{4}$ ... | ... SEŘAZENÁ $\frac{1}{4}$ (B) ... | ... SEŘAZENÁ $\frac{1}{2}$ (A) ... |

Tabulka 2: Na slití A a B nemáme dostatečný pracovní prostor

Klíčová myšlenka spočívá v tom, že místo abychom seřadili druhou polovinu neseřazeného pole, seřadíme tu první, čímž umístíme pracovní prostor mezi dvě seřazené části pole. Pracovní prostor se pak bude překrývat se seřazenou polovinou pole A.

Nyní uvažujme dva extrémní případy, které mohou nastat:

1. Všechny prvky v B jsou menší než prvky v A. Potom slívací algoritmus postupně přesune B do pracovního prostoru. Protože jejich velikosti jsou stejné, vše je v pořádku.

2. Všechny prvky v B jsou větší než v A. V tomto případě slívací algoritmus postupně přesouvá prvky z A do pracovního prostoru. Po zaplnění celého původního pracovního prostoru začne algoritmus přepisovat první polovinu pole A. Nejedná se ale o neslité prvky, takže podle druhé podmínky je vše v pořádku. Pracovní pole se tak tedy přesune na pravou stranu. Nyní algoritmus začne prohazovat prvky z B s pracovním prostorem. Pracovní pole se tak tedy přesune na levou stranu (viz tabulka 3).

| ... SEŘAZENÁ $\frac{1}{4}$ (B) ... | ... PRACOVNÍ PROSTOR $\frac{1}{4}$ ... | ... SEŘAZENÁ $\frac{1}{2}$ (A) ... |

$\Downarrow$

| ... PRACOVNÍ PROSTOR $\frac{1}{4}$ (B) ... | ... SLITÉ $\frac{3}{4}$ ... |

Tabulka 3: Slití A a B

Takto algoritmus pokračuje a rekurzivně zmenšuje a postupně slévá seřazená pole. Když se velikost pole zmenší pod určitou mez, můžeme přepnout na jiný řadící algoritmus (například Insertion sort) a tím optimalizovat rychlost výsledného algoritmu.

2 Optimalizovaná verze

2.1 Provedené optimalizace

Algoritmus in-place merge sort je rekurzivní, tedy postupně dělí pole na menší a menší, která řadí a následně slévá dohromady. Celková asymptotická složitost tohoto algoritmu je $O(n \log n)$. Vzhledem k tomu, že asymptotická složitost nebere v úvahu konstanty, může být pro malá n ve skutečnosti výhodnější použít algoritmus s vyšší asymptotickou složitostí. Můžeme si například představit, že skutečná složitost algoritmu A je $100\,000 * n \log n$ a algoritmu B $0.1 * n^2$. Asymptoticky je tedy lepší algoritmus A, avšak například pro $n = 10$ bude zcela jistě výhodnější použít algoritmus B.

Z výše uvedeného důvodu jsem algoritmus in-place merge sort zoptimalizoval použitím algoritmu Insertion sort v určité fázi rekurzivního dělení vstupního pole. Insertion sort má asymptotickou složitost $O(n^2)$, ale má oproti rekurzivnímu merge sortu menší nároky na režii.

Důležitou otázkou bylo, jak zvolit hranici h pro ukončení rekurzivního volání a spuštění algoritmu Insertion sort. V tabulce 4 jsou zaznamenány změřené časy pro různé hodnoty h. Měření probíhalo pro náhodnou, předem vygenerovanou vygenerovanou posloupnost délky 100 000. Všechny běhy programu měly tatáž vstupní data. Pro zajímavost je na konci uveden naměřený čas pro klasický merge sort. Data potvrzují správnost úvahy výše, tedy že pro malá pole se v tomto případě vyplatí použít asymptoticky horší algoritmus Insertion sort.

h	Algoritmus	Naměřený čas `[ms]`
1	In-place Merge sort	78,69
2	Hybridní in-place Merge sort	75,34
5	Hybridní in-place Merge sort	75,04
6	Hybridní in-place Merge sort	74,71
10	Hybridní in-place Merge sort	77,71
20	Hybridní in-place Merge sort	86,72
-	Klasický Merge sort	28,19

Tabulka 4: Naměřené časy pro různá h, náhodné pole délky 100 000

2.2 Naměřené časy

V tabulce níže jsou zaznamenány časy různých implementací.

Algoritmus / Velikost vstupu	100 000	500 000	1 000 000	10 000 000
In-place Merge sort	78,69 ms	476,71 ms	1,03 s	13,716 s
Hybrid in-place Merge sort ($h = 6$)	74,71 ms	456,55 ms	0,979 s	12,844 s
Classic Merge sort	28,19 ms	151,11 ms	0,31 s	3,382 s

Tabulka 5: Naměřené časy pro náhodnou posloupnost

Algoritmus / Velikost vstupu	100 000	500 000	1 000 000	10 000 000
In-place Merge sort	56,43 ms	343,49 ms	751,58 ms	10,382 s
Hybrid in-place Merge sort ($h = 6$)	45,4 ms	284,96 ms	628,215 ms	8,222 s
Classic Merge sort	14,89 ms	82,91 ms	173,015 ms	1,997 s

Tabulka 6: Naměřené časy pro obráceně seřazenou posloupnost

2.3 Závěr

Z naměřených časů je zřejmé, že použití algoritmu Insertion sort pro menší pole zrychlilo celkový algoritmus. Celkové zrychlení je závislé na zvolení meze h pro spuštění algoritmu Insertion sort. Z měření nejlépe vyšla verze algoritmu pro $h = 6$. Tato hodnota je tedy uvedena v tabulkách a je použita pro další měření.

Pro náhodnou posloupnost je naměřené zrychlení kolem 6 %, ale pro obráceně seřazenou posloupnost se algoritmus zrychlil o více než 20 %. Při velikosti vstupu 10^7 se jedná o více než dvě vteřiny, což je významný rozdíl.

Paměťová složitost $O(1)$ in-place verze algoritmu Merge sort je však vykoupena složitostí časovou. Ve všech testech byl klasický Merge sort několikrát rychlejší než optimalizovaná in-place verze. In-place verze provadí mnohem více prohození prvků, což se odráží právě na výsledné časové složitosti. Z tohoto důvodu postrádá uplatnění v praxi, kde lineární paměťová složitost není takový problém.

Reference

[1] Xinyu LIU, Larry *Elementary Algorithms* [online]. Dostupné z: `http://bit.ly/2g39dc4`