KIV / PRO
Nový sort algoritmus pro SQL Query

Michal Malík

Zadání

Najděte v anglicky psaném odborném časopise článek v délce alespoň 5 stránek o nějakém algoritmu řešícím libovolný problém. Algoritmus popište do českého referátu tak, aby ho podle vašeho názoru pochopil běžný student 2. ročníku informatiky. Váš text musí svědčit o tom, že algoritmu rozumíte, a musí ho z něj pochopit i nezasvěcený čtenář, důležité informace musí zůstat zachovány i ve vašem přepisu.

Analýza problému

Řazení je důležitou komponentou ve většině databázových systémů (DBMS) a bývá jak výkonně, tak paměťově náročné. Výkonnost DBMS query je z velké části právě ovlivněná řadícími algoritmy, kde zároveň chceme také zachovat stabilitu. [1] Stabilní řadící algoritmus je takový algoritmus, který zachovává vzájemné pořadí prvků se stejným klíčem. Pokud byl prvek A v původní (neseřazené) posloupnosti před prvkem B (se stejným klíčem), pak v seřazené posloupnosti bude prvek A stále před prvkem B.
Externí třídicí algoritmy pro práci s velkými daty obvykle probíhají ve dvou fázích. V první fázi se vytvářejí uspořádané dílčí soubory (tzv. „runs“) a ve druhé se tyto soubory slučují do jednoho výsledného seřazeného výstupu. [2] Oblíbenou skupinou jsou slučovací algoritmy **(Merge-Based Sorting)**, které rozdělí vstupní data na přibližně stejně velké bloky, každý seřadí v paměti a uloží na disk. Poté se tyto bloky postupně slučují.

Navržený algoritmus

Navrhovaný algoritmus vychází z metody zmíněné v předchozí sekci.
Hlavní cíl našeho postupu je rozdělit elementy $a_i, i = 1, ..., n$ z pole dat A o velikosti n do malých segmentů $\sigma_j, j = 1, .., m$ kde m je počet rozdělení. Všechny tyto rozdělení mají stejnou velikost l, jenž získáme pomocí vzorečku $l = \frac{n}{m}$. [3]
Všechny segmenty budou zpočátku prázdné, a každému z nich vytvoříme ukazatel $last_j$ na poslední hodnotu daného segmentu, kterému přiřadíme základní hodnotu 0. Ta reprezentuje poslední vložený prvek do segmentu σ_j. Prvek a_i můžeme vložit do segmentu následujícími dvěma způsoby:

$$a_i >= last_i, \text{ kde } j = 1, ., m. => a_i \text{ přiřadíme do segmentu } \sigma_j$$

$$a_i < last_j, \text{ kde } last_j \neq l => a_i \text{ přiřadíme do segmentu } \sigma_{temp}$$

Pakliže naplníme dočasný segment σ_{temp}, tak pro m segmentů započneme m-násobné sjednocení. Tím získáme jedno velké seřazené pole obsahující všechno, co bylo dosud v těchto segmentech uloženo. Pak provedeme 2-násobné sjednocení s daným celkem a segmentem σ_{temp}. Z tohoto nového celku o velikosti L znovu spořádaně vytvoříme S segmentů, kde $S = \frac{L}{l}$. Algoritmus bude poté pokračovat ale pouze se segmenty $S + 1$ do m.

Listing 1: Navržený algoritmus pro třídění na základě segmentů

```
# Inicializace
sigma = [[] for j in range(1, m+1)]      # m segmentu sigma_j
last = [0 for j in range(1, m+1)]        # ukazatele last_j
sigma_temp = []                          # docasny segment
last_temp = 0
i = 1                                    # index aktualniho prvku
b = 1                                    # pocatecni buffer

while i <= n:
    a_i = A[i]                           # nacti prvek z vstupniho pole
    j = b
    append = False

    while not append and j <= m:
        if a_i >= sigma[j][-1] and last[j] != l:
            last[j] += 1
            sigma[j].append(a_i)
            append = True
        else:
            j += 1

    if append:
        append = False
        i += 1
        continue
    else:
        # vloz prvek do docasneho segmentu na spravne misto
        insert_sorted(sigma_temp, a_i)
        last_temp += 1
        i += 1

        if last_temp > l:
            merged = multiway_merge(sigma[b:m+1])   # (m-b+1)-nasobne slucovani
            merged = two_way_merge(merged, sigma_temp)
            L = len(merged)

            # rozdel sloucena data zpet do bufferu
            while L > l:
                sigma[b] = merged[:l]
                merged = merged[l:]
                L -= l
                b += 1
            sigma[b] = merged[:L]
            sigma_temp = []
            last_temp = 0

output = multiway_merge(sigma)
```

Lemma : Nechť A je pole s n elementy. Stable-Sort algoritmus garantuje, že výsledné pole je stabilní. [3]

Sjednocení

Sjednocení je proces, při kterém se už dvě seřazené pole zkombinuje do jednoho seřazeného seznamu.
Nejznámějším a nejvíce používaným způsobem dvou polí A s prkvy n_1 a B s elementy n_2 je pomocí porovnávání od nejmenších prvků po největší, kde pokaždé vybereme ten menší prvek a vložíme jej do našeho nového sjednoceného pole. Sjednocovací algoritmus v tomto případě provede $n_1 + n_2 - 1$ porovnávání a $n_1 + n_2$ přesunů dat do nového pole. Pořadí, ve kterém jsou jednotlivé hodnoty ze seznamů A a B vybírány a zapisovány do výstupního seznamu C, je určeno jejich indexem. Tímto způsobem lze slučování snadno provést stabilně – tzn. zachovat původní pořadí prvků se stejnou hodnotou.

	A		B
1	1	1	2
2	2	2	7
3	6	3	7
4	11	4	9
5	15	5	11
6	17	6	15
7	17	7	17

Figure 1.a

	C
1 from A_1	1
2 from B_1	2
3 from A_2	2
4 from A_3	6
5 from B_2	7
6 from B_3	7
7 from B_4	9
8 from A_4	11
9 from B_5	11
10 from A_5	15
11 from B_6	15
12 from A_6	17
13 from A_7	17
14 from B_7	17

Figure 1.b

Obrázek 1: Grafická reprezentace sjednocení

Stabilita znamená, že pokud se při slučování objeví dva prvky se stejnou hodnotou – jeden ze seznamu A a druhý ze seznamu B – dostane přednost prvek z A. Zároveň se u všech prvků se stejnou hodnotou zachová jejich původní pořadí po dokončení slučování.
Navrhovaný m-násobný slučovací algoritmus (m-way merge) může být implementován dvěma způsoby:

1. buď jako série 2-násobných slučování (tj. slučování dvou seznamů po sobě) s dodržením podmínky stability
2. nebo jako současné slučování všech m seznamů najednou.

Běžný 2-way merge lze tedy zobecnit na m seřazených seznamů, kde m je celé číslo větší než 2. Abychom během slučování dokázali sledovat, který prvek je momentálně nejmenší v každém seznamu, budeme potřebovat m ukazatelů, aby každý segment měl svůj ukazatel.
Počet porovnání potřebných pro sloučení m seřazených seznamů, z nichž každý obsahuje l prvků, je:

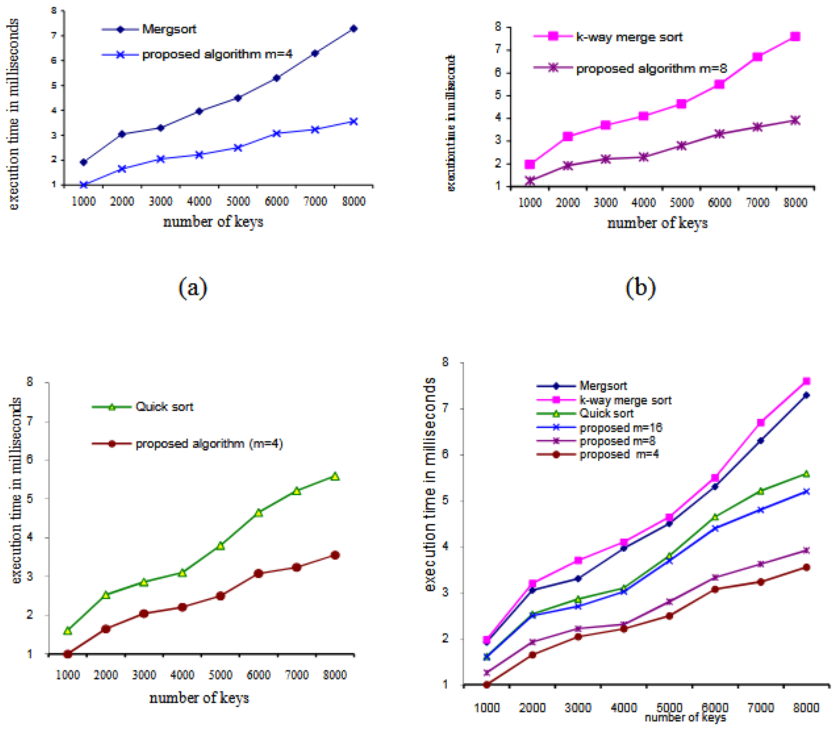
$$O(n \log_2(m)), \quad \text{kde } n = m \cdot l \tag{1}$$

tedy složitost roste úměrně s počtem prvků a logaritmicky s počtem seznamů.

Výsledky

Časová složitost

Procházení skrze počáteční datové pole A a jeho dělení na m podsegmentů je provedeno v lineárním čase. Jediná výjimka je v případě, kdy dočasný segment σ_{temp} se naplní. Proto můžeme provedení v nejlepším čase brát v tu chvíli, kdy se dočasný segment nikdy nenaplní.
Navržený stabilní třídicí algoritmus vyžaduje v průměru složitost $O(n \log m)$ porovnání a $O(n)$ přesunů prvků. V nejhorším případě může každý segment obsahovat pouze jeden prvek, takže je nutné opakovaně vkládat prvky do dočasného pole, dokud se nenaplní.
Předpokládejme, že $n = m \cdot r$ pro celé číslo $r > 0$. Pokud $r = 1$ (tedy počet bufferů m se rovná počtu záznamů n), má algoritmus v nejhorším případě složitost $O(n \log n)$. S klesajícím počtem bufferů (rostoucím r) se náklady snižují, ale jen do určité meze. Z výsledků (obr. 2) vyplývá, že nejvhodnější hodnota m je 4.



Obrázek 2: Navržený algoritmus v porovnání s ostatními řadícími algoritmy

Paměťová složitost

Algoritmus vyžaduje pomocné úložiště v podobě segmentů, kde jejich celková velikost odpovídá velikosti rovné vstupnímu poli.

Závěr

V tomto článku nám byl představen nový řadící algoritmus, který využívá sjednocovacího procesu a je schopen zachovat stabilní pořadí prvků. Na grafech s výsledky jsme si dokázali, že Stablesort může konkurovat více známým algoritmům se svojí rychlostí $O(n \log m)$, kde m je počet segmentů, na které si rozdělíme naše vstupní data.

Odkazy

[1] D. E. Knuth, *Sorting and Searching* (The Art of Computer Programming). Reading, MA: Addison-Wesley Publishing Company, 1973, sv. 3.

[2] G. Franceschini a V. Geffert, „An In-Place Sorting with O(n log n) Comparisons and O(n) Moves," in *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, IEEE, 2003, s. 242–250.

[3] H. I. Mathkour, „A NEW SORTING ALGORITHM FOR ACCELERATING JOIN-BASED QUERIES," *Mathematical and Computational Applications*, roč. 15, č. 2, s. 208–217, 2010.