

Externes Sortieren

VO Algorithm Engineering

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

13. VO

6.12.2005

Literatur für diese VO

- U. Meyer, P. Sanders und J. Sibeyn (Eds.), Algorithms for Memory Hierarchies, Advances Lectures, Lecture Notes in Computer Science 2625, Springer 2003:
 - Kapitel 2: R. Pagh: Basic External Memory Data Structures, S. 14-35
 - Kapitel 3: A. Maheshwari und N. Zeh: A Survey of Techniques for Designing I/O-Efficient Algorithms

2

Überblick

- Einführung Externes Sortieren
- Externes Merge-Sort
- Untere Schranke für Externes Sortieren

3

Wichtigste Externe Sortierparadigmen

Distribution Paradigma

1. **Partitionierungsphase:** Partitioniere die Input-Folge S in Teilfolgen S_0, \dots, S_k , so dass für alle $0 \leq i < j \leq k$ und zwei beliebige Elemente $x \in S_i$ und $y \in S_j$: $x \leq y$. Dazu wähle Splitters $x_1 \leq \dots \leq x_k$ von S .
2. **Sortierphase:** Sortiere jede Teilmenge S_i rekursiv.
3. **End:** Hänge die sortierten Teilfolgen aneinander.

- Beispiel Quicksort:
 - Falls alle Teilmengen $O(|S|/k)$ Elemente besitzen, dann Laufzeit: $O(N \log N)$

4

Wichtigste Externe Sortierparadigmen

Merging Paradigma

1. **Run-Formationsphase:** Partitioniere die Input-Folge in sortierte Teilsequenzen: „Runs“
2. **Merging-Phase:** Verschmelze diese „Runs“ solange, bis nur noch ein „Run“ existiert

- Beispiel Merge Sort:
 - Die Runs sind zu Beginn 1-elementig
 - Verschmelzen mit 2-Way Merging (in Paaren aufgeteilt).
 - Laufzeit: $O(N \log N)$

5

Externer Merge-Sort

I/O-Komplexität des internen Merge-Sort:

1. **Run-Formationsphase:** 0 I/Os
2. **Merging-Phase:**
 - Verschmelzen der Teilfolgen S_1 und S_2 : $O(1 + (|S_1| + |S_2|)/B)$ I/Os
 - Anzahl der I/Os auf einer Schicht: $O(N + N/B)$ I/Os
 - Über alle Schichten: $O((N + N/B) \log N)$ I/Os

6

Externer Merge-Sort

1. Verbesserung: Verhindere in der Run-Formationsphase 1-elementige Mengen!

- Beende die Aufteilung bei Teilmengen der Länge M
 - Lade die N/M Stücke nacheinander in das Main Memory
 - Sortiere diese im Main Memory
 - Schreibe die sortierte Teilfolge zurück nach EM
- Diese Aufteilung kostet zusätzlich: $O((N/M)(M/B))=O(N/B)$ I/Os für das Hin- und Herkopieren (das Sortieren an sich kostet kein I/O)
- Anzahl der I/Os auf einer Schicht: $O(N/M+N/B)$ I/Os
- Über alle Schichten: $O((N/M+N/B) \log(N/M))$ I/Os
- Dies ist gleich $O(N/B(1+\log(N/B)))$

7

Externer Merge-Sort

2. Verbesserung: Verschmelze jeweils $k=M/(2B)$ Runs

- Kopiere die jeweils kleinsten Elemente x_1, \dots, x_k der Runs S_1, \dots, S_k in das MM
 - Kopiere das Minimum x_i der Elemente in den Output Run
 - Lese das nächste Elemente von S_i usw
- Verschmelzen der Teilfolgen S_1, \dots, S_k : $O(k+(|S_1|+\dots+|S_k|)/B)$ I/Os
- Anzahl der I/Os auf einer Schicht: $O(N/M+N/B)$ I/Os
- Über alle Schichten: $O((N/M+N/B) \log_{M/B}(N/B))$ I/Os
- Dies ist gleich $O(N/B(1+\log_{M/B}(N/B)))$

8

Externer Merge-Sort

Effiziente Verschmelzung von k Runs? (Intern)

Minimumsuche der Keys x_1, \dots, x_k ?

- Naiv: $O(k)$: zu teuer, denn dann würde die Laufzeit der Merge-Phase $O(k N \log_k(N/B))$ sein.
- Lösung: Halte die jeweils kleinsten Elemente x_1, \dots, x_k in einer Prioritätsschleife in MM; Speicherplatz: $O(k)=O(M/B)$

9

Laufzeit von k -Multiway Merging

- Phase 1: Sortiere N/M Stücke der Länge M :
 $O((N/M) M \log M) = N \log M$
 - pro Schicht: Verschmelze N Elemente inkl. Minimumsuche:
 $O(N \log k)$
 - Tiefe des Baumes: $O(\log_k(N/B))$
 - Insgesamt: $O(N \log M) + (N \log k) \log_k(N/B) = O(N \log N)$
- Verwende Logarithmus-Gesetz: $\log_b a = (\log_c a) / (\log_c b)$

10

Externes Sortieren

- Theorem: Eine Menge von N Elementen kann mit Hilfe von k -Multiway Merging in interner Zeit $O(N \log N)$ mit $O(N/B(1+\log_{M/B}(N/B)))$ I/Os sortiert werden.

11

Untere Schranken für Sortieren

- Wieviele I/O-Operationen sind mindestens notwendig um eine gewisse Permutation der Eingabefolge zu erhalten?
- Soviele sind auch notwendig um diese Folge zu sortieren.

12

Untere Schranken für Sortieren

Restriktionen unseres Modells:

1. MM kann M Elemente aufnehmen.
2. EM ist ein Array von Elementen.
3. EM kann nur an Blockbegrenzungen beschrieben werden
4. Zu Beginn enthalten die ersten N/B Blöcke des EM den Input
5. Alles andere ist leer (MM und restl. EM)
6. Nach Ablauf enthalten die ersten N/B Blöcke im EM den Output
7. Einzige erlaubte Operation: move (nicht split, duplicated)
8. read: bewegt B Elemente eines Blocks von EM nach MM
9. write: bewegt B Elemente von MM in einen Block von EM

13

Untere Schranken für Sortieren

- Theorem: Um eine gegebene Permutation von N Elementen herzustellen wird mindestens die folgende Anzahl von I/Os benötigt:

$$t \geq 2 \frac{N \log(N/eB)}{B \log(\epsilon M/B) + 2 \log(N/B)/B}$$

- Für $N=O((\epsilon M/B)^{B/2})$ dominiert der Term $\log(\epsilon M/B)$ den Nenner und wir erhalten:

$$2 \frac{N \log(N/eB)}{B O(\log(\epsilon M/B))} = \Omega\left(\frac{N}{B} \log_{M/B} \frac{N}{B}\right)$$

14

Beweis: Untere Schranken für Sortieren

- Finde obere Schranke c_t für die Anzahl an verschiedenen Permutationen, die nach t I/Os generiert werden können.
- Da es $N!$ viele verschiedene Permutationen gibt, muss t groß genug sein, so dass $c_t \geq N!$
- Diese Ungleichung wird nach t aufgelöst.

15

Beweis: Untere Schranken für Sortieren

- Ein Zustand eines Algorithmus kann abstrakt folgendermaßen beschrieben werden:
 - die Menge der Elemente im MM
 - die Menge der Elemente in jedem nicht-leeren Block in EM
 - die Permutation in der die Elemente in jedem nicht-leeren Block in EM gespeichert sind.
- Wir nennen 2 Zustände äquivalent, wenn sie in den ersten beiden Komponenten übereinstimmen.
- Im Endzustand befinden sich die Elemente in N/B Blöcken von jeweils B Elementen im EM.
- Jede Äquivalenzklasse des Endzustands besteht aus $(B!)^{N/B}$ Zuständen.

16

Beweis: Untere Schranken für Sortieren

- Es genügt also zu untersuchen, wann die Anzahl der Äquivalenzklassen der Endzustände C_t , die nach t I/Os erreicht werden können, die Zahl $N!/(B!)^{N/B}$ übersteigt.

Wir schätzen C_t induktiv: $C_0=1$.

Lemma:
$$C_{t+1} \leq \begin{cases} C_t N/B, & \text{falls die I/O-Operation ein read ist} \\ C_t N/B^{(M/B)}, & \text{falls die I/O-Operation ein write ist} \end{cases}$$

Beweis: Ein read spezifiziert welcher der N/B nicht-leeren Blöcke gelesen wird. Ein write zusätzlich, welche Elemente geschrieben werden und in welcher Permutation. Es gibt i aus M (kleiner gleich B aus M) Möglichkeiten für die Auswahl, die Permutation ist irrelevant bzgl. Äquivalenzklassen.

17

Beweis: Untere Schranken für Sortieren

Lemma: In jedem Algorithmus, der Permutationen in dem hier beschriebenen Modell generiert, ist die Anzahl der read-Operationen gleich derjenigen der write-Operationen.

Beweis: Ein read vergrößert die Anzahl der leeren Blöcke in EM, ein write verringert diese. Zu Beginn und am Ende gibt es genau N/B nicht-leere Blöcke in EM. Also ist die Anzahl der read und write Operationen gleich.

18

Beweis: Untere Schranken für Sortieren

Für gerade t erhalten wir aus den beiden Lemmas:

$$\frac{N!}{(B!)^{N/B}} \leq C_t \leq \left(\frac{N}{B}\right)^t \binom{M}{B}^{t/2}$$

Wir benutzen die Formeln $(m/e)^m \leq m! \leq m^m$:

$$\binom{M}{B} \leq M^B / B! \leq (eM/B)^B$$

$$N! / (B!)^{N/B} \geq (N/e)^N / B^N$$

Deswegen folgt aus obiger Formel:

$$\left(\frac{N}{B}\right)^t \left(\frac{eM}{B}\right)^{Bt/2} \geq \left(\frac{N}{eB}\right)^N$$

Nach Auflösen nach t erhält man die Formel des Theorems. ENDE