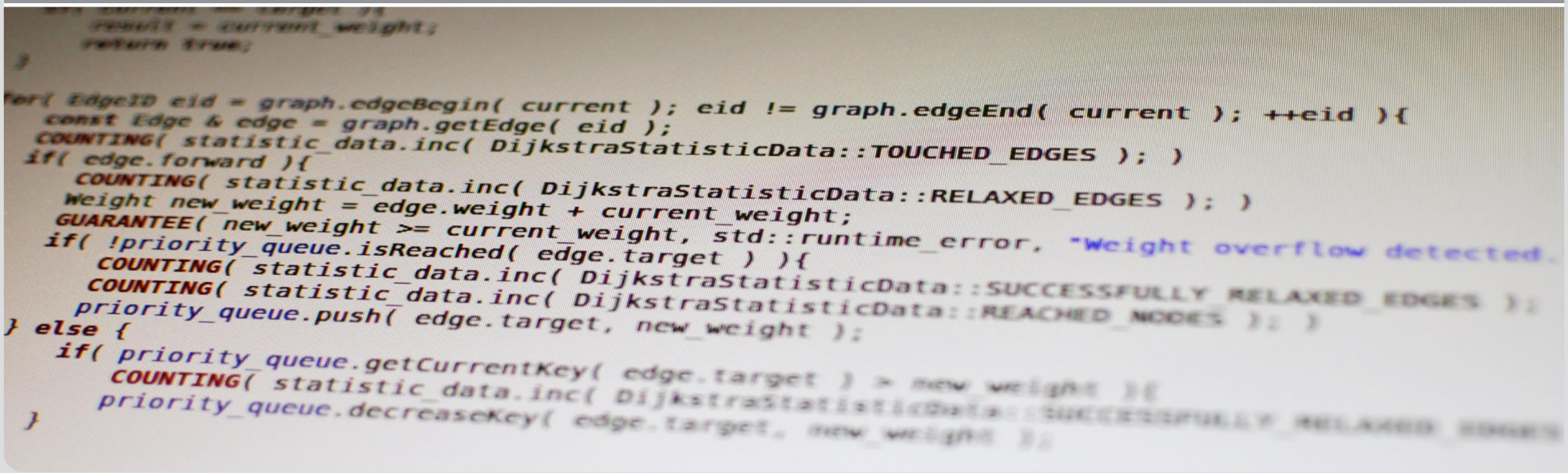
Karlsruher Institut für Technologie

Übung 7 – Algorithmen II

Moritz Laupichler, Hans-Peter Lehmann – *{moritz.laupichler, hans-peter.lehmann}@kit.edu*

`http://algo2.iti.kit.edu/AlgorithmenII_WS22.php`

Institut für Theoretische Informatik - Algorithmik II



Themenübersicht

Übungsinhalt

- External Memory Algorithmen

External Memory
Speichermodell

Speichermodell

Latenzen

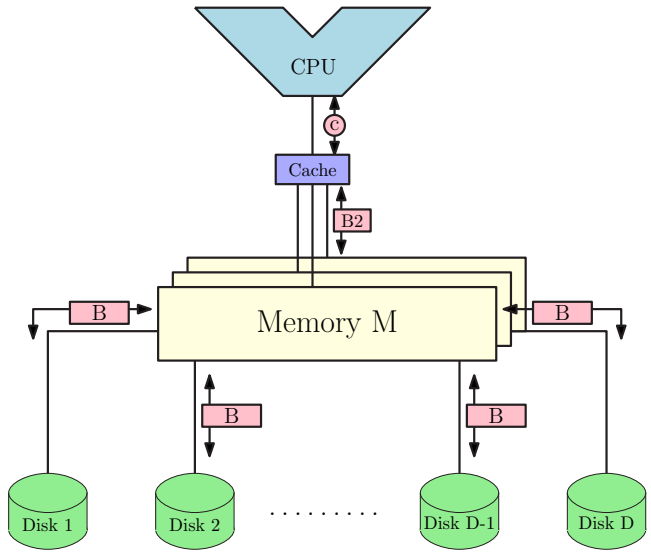
1 CPU cycle	0.3 ns	1 s
Level 1 cache access	0.9 ns	3 s
Level 2 cache access	2.8 ns	9 s
Level 3 cache access	12.9 ns	43 s
Main memory access	120 ns	6 min
Solid-state disk I/O	50-150 µs	2-6 days
Rotational disk I/O	1-10 ms	1-12 months
Internet: SF to NYC	40 ms	4 years
Internet: SF to UK	81 ms	8 years
Internet: SF to Australia	183 ms	19 years
OS virtualization reboot	4 s	423 years
SCSI command time-out	30 s	3000 years
Hardware virtualization reboot	40 s	4000 years
Physical system reboot	5 m	32 millenia

Speichermodell

- Vitter und Shriver:
  - Parallel Disk Model (PDM)

- Speicherzugriffe in Blöcken
- Blockzugriffe minimieren $\rightarrow$ Datenlokalität

- Muster wiederholt sich in Speicherhierarchie immer wieder



Blockgrößen

Einflussfaktoren

- T_{seek}: Positionierungszeit
- W_{max}: maximale Bandbreite

$\rightarrow$ Lesedauer: $T = T_{seek} + B / W_{max}$

Optimale Blockgröße

- Beispiel: RAM
  - $T_{seek} = 30$ ns (Zykluszeit)
  - $W_{max} \approx 25$ GB/s
  - Anzahl Blöcke pro Zeile: $R = 128$
  - Ziel: 95% Auslastung der Bandbreite bei sequenziellem Lesen

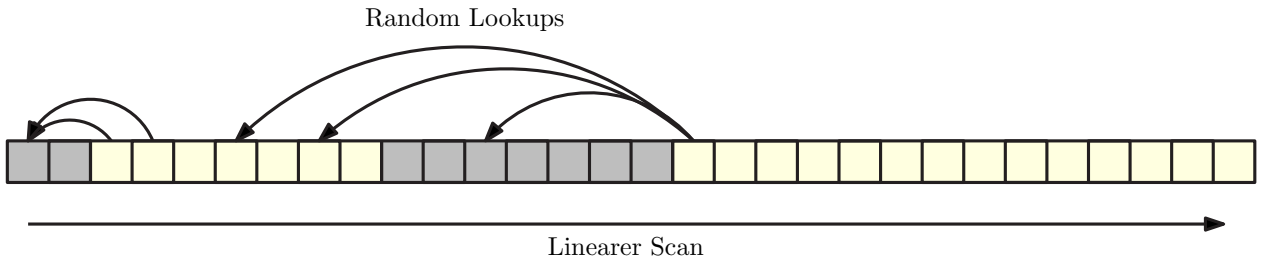
    $\rightarrow W = \frac{B}{T} = B / \left(\frac{1}{128} T_{seek} + B / W_{max} \right) \stackrel{!}{=} 0.95 \cdot W_{max}$

    $\rightarrow B = 0.15 \cdot W_{max} \cdot T_{seek} = 111$ Byte!

I/O-effizientes Design

I/O-effizientes Design

- nicht nur relevant bei Disk-I/O
- Beispiel: kürzeste Wege-Bäume auf großen Straßennetzen
  (z.B. Europa)
  - Dijkstra (annähernd linear): $\approx 3 - 5$ Sek.
  - Breitensuche (untere Schranke?): ≈ 2 Sek.
  - PHAST (linearer Scan): < 0.2 Sek.



- Strukturierter Zugriff als wichtiges Designprinzip

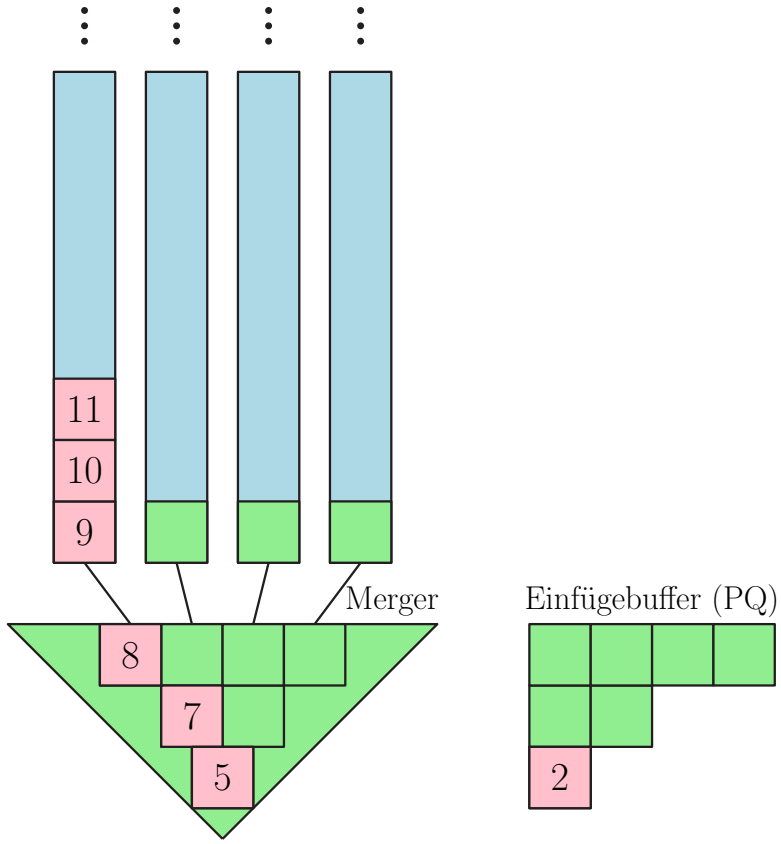
I/O-effizientes Design

Grundlegende Techniken

- Zugriffsmuster
  - Random Access erwartet $\mathcal{O}(n)$ I/Os
  - Linearer Scan $\mathcal{O}(n/B)$ I/Os
- Stack / Queue
- Sortieren
  - $\mathcal{O}\left(\frac{2n}{B}\left(1+\lceil\log_{M/B}\frac{n}{M}\rceil\right)\right)$ I/Os
  - lokale Kriterien
  - oft vorbereitend für linearen Scan
- Prioritätswarteschlangen
  - $\approx \text{sort}(n)$ I/Os
  - nutzbar als Warteliste: „Speicherzugriff auf später verschieben“

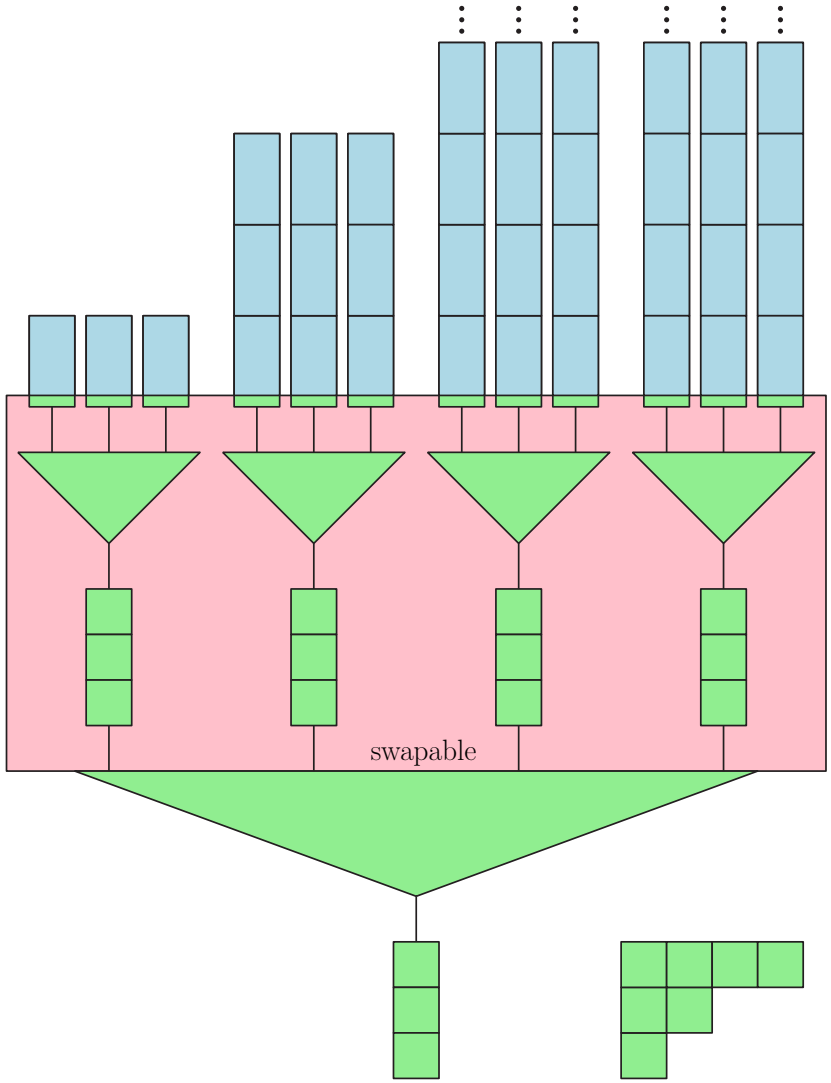
Externe Priority Queue

- sortierte Teilsequenzen
- Einfügepuffer (interne PQ)
- Merger (interne PQ)
- Operationen:
  - `Insert`
  - `DeleteMin`
- natürliches Limit:
  - Anzahl eingefügter Elemente beschränkt
- Was tun bei mehr Elementen?



Externe Priority Queue

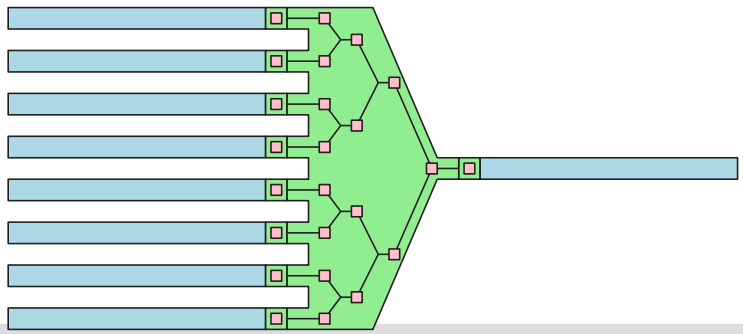
für sehr große Datenmengen



Externes Sortieren

Zwei-Phasen Algorithmus

- *Run Formation*
  - Run entspricht einem Teilbereich zu sortierender Daten
  - $\lceil \frac{n}{M} \rceil$ Stück, Größe M
  - Jeweils $\mathcal{O}(M \log M)$ Arbeit (Sortieren)
  - Alle Daten einmal lesen + schreiben
- *Multiway Merge*
  - Jede Mischphase liest und schreibt alle Daten $\rightarrow \frac{2n}{B}$ I/Os
  - Hilfsmittel: Interne PQ für $\frac{M}{B}$ Eingabeströme
  - Pro Phase: Gruppen von $\frac{M}{B}$ Runs zu einem Run mergen $\rightarrow \lceil \log_{M/B} \frac{n}{M} \rceil$ Phasen
- Innere Arbeit:
  $\mathcal{O}\left(\frac{n}{M} \cdot M \log M + n \log \frac{M}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$
- I/O Operationen:
  $\mathcal{O}\left(\frac{2n}{B} + \frac{2n}{B} \cdot \lceil \log_{M/B} \frac{n}{M} \rceil\right)$



Klausuraufgabe

Nachklausur 2022

Gegeben sei ein aufsteigend sortiertes Array A aus n unterschiedlichen Ganzzahlen $\in [0, U]$. Die Operation *predecessor*(x) gibt die größte Zahl im Array zurück, die noch kleiner ist als x. Die Operation *predecessorIndex*(x) gibt den Array-Index zurück, an dem diese Zahl steht. Für Parameter kleiner als alle Array-Einträge sind die beiden Funktionen undefiniert. In dieser Aufgabe beginnen Array-Indizes bei 1.

Gegeben sei das Array $A = [1, 4, 8, 10]$. Geben Sie die Werte der folgenden Operationen an.

predecessor$(6) =$
predecessorIndex$(7) =$

Klausuraufgabe

Nachklausur 2022

Der Arbeitsspeicher sei in Seiten der Größe B aufgeteilt. Zusätzlich gibt es einen Cache, der deutlich schneller als der Arbeitsspeicher zugegriffen werden kann. Beim ersten Zugriff auf eine Seite liegt diese noch nicht im Cache (Cache-Fault), wodurch der Zugriff langsam ist. Danach kann deutlich schneller auf die Seite zugegriffen werden. Gehen Sie im Folgenden davon aus, dass vor jeder Anfrage an Ihre Datenstruktur der Cache geleert wird.

Das Array A sei linear im Arbeitsspeicher abgelegt. Beschreiben Sie einen Algorithmus, der *predecessor* und *predecessorIndex* für eine Zahl mit $\mathcal{O}(\log n)$ Cache-Faults bestimmen kann. Begründen Sie die Anzahl der Cache-Faults kurz.

Klausuraufgabe

Nachklausur 2022

Beschreiben Sie eine Datenstruktur, die *predecessor* und *predecessorIndex* für eine Zahl mit $\mathcal{O}(\log_B n)$ Cache-Faults bestimmen kann. B sei hier eine Eingabegröße, also im $\mathcal{O}$-Kalkül nicht irrelevant. Zur Vereinfachung sei n eine Potenz von B, also $\log_B n \in \mathbb{N}$. Die Datenstruktur darf maximal $\mathcal{O}(n)$ Platz benötigen. Begründen Sie den Platzbedarf und die Anzahl der Cache-Faults kurz.
Hinweis: $\sum_{k=0}^{a} q^k = \frac{1-q^{a+1}}{1-q}$ für $q \neq 1$.

Ende!

Feierabend!