SWE3 HÜ 2: External MergeSort

Erik Pitzer <erik.pitzer@fh-hagenberg.at>

Medizin- und Bio-Informatik – WS 2024/25

Name

Aufwand (in h)

Punkte

Aufgabe	Lösungsidee	Implementierung	Tests	Gesamtpunkte
1	30% / 12P	20% / 8P	50% / 20P	40
2	20% / 12P	50% / 30P	30% / 18P	60

Aufgabe 1: Merge Sort Dokumentieren

Vervollständigen Sie die Implementierung und Dokumentation, fügen Sie mehr Testfälle hinzu und überlegen Sie eine alternative ASCII-Graphik Visualisierungen um den Aufbau des Heaps und die anschließende Sortierung besser zu veranschaulichen.

Aufgabe 2: External Merge Sort

Implementieren Sie eine Klasse `merge_sorter`, die eine Methode `sort_on_disk` anbietet. Dieser Methode wird der Name einer zu sortierenden Datei übergeben. Die Datei soll so sortiert werden, dass minimaler Hauptspeicher verbraucht wird. Insbesondere solle es möglich sein sehr große Dateien (mehrere hundert MB) zu sortieren.

Dazu sollen, wie in der Übung besprochen zuerst die Eingabedatei auf zwei Dateien aufgeteilt werden. Von diesen zwei Dateien werden dann immer größere sortierte chunks zusammengefügt und abwechselnd in zwei weitere Dateien geschrieben, wie in den Abbildungen 1 und 2.

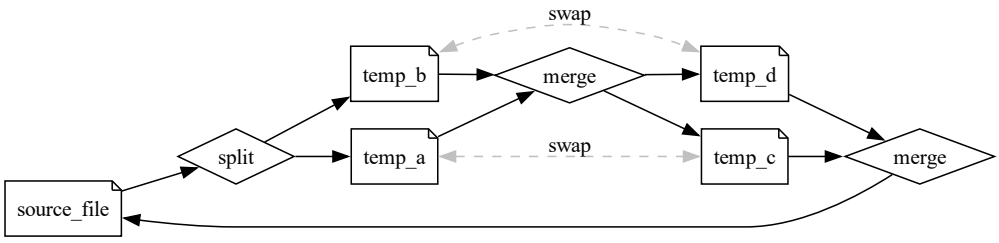

Figure 1: Aufteilung und wiederholtes zusammenführen von immer größeren Chunks.

Ein paar Implementierungshinweise:

1. Die Klasse `merge_sorter` soll die in der Übung besprochene Klasse `file_manipulator` für alle Dateioperationen verwenden. Diese Dateioperationen könnten sein: eine Datei kopieren, eine Datei mit Zufallswerten füllen, eine Datei in mehrere Dateien aufsplitten, den Inhalt einer Datei ausgeben.

2. Die Klasse `file_manipulator` operiert auf `ifstreams` und `ofstreams`. Die einzigen erlaubten Operationen auf diese Streams sind nur `<<` und `>>`.

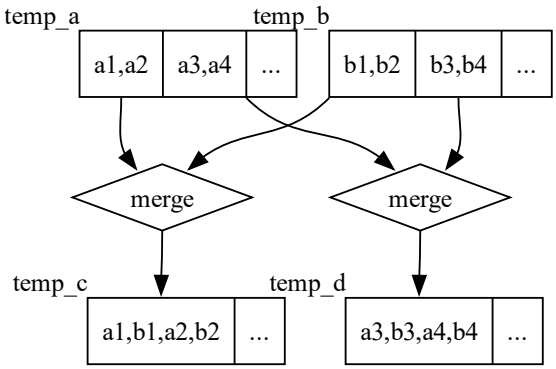

Figure 2: Jeweils Chunks der Größe n von A und B, werden zu einem Chunk der Größe $2n$ zusammengeführt. Die neuen Chunks werden abgechselnd in C und D geschrieben.

3. Die Klasse `stream_reader<value_type>` kann zu Hilfe genommen werden um Streams von Tokens zu lesen und ein Vorschau auf das nächste Token zu bekommen.
   - Der Konstruktor bekommt entweder einen bereits bestehenden (File-)Stream oder einen Dateinamen von dem gelesen werden soll.
   - Mit `has_next()` muss geprüft werden ob ein nächstes Token zur Verfügung steht.
   - Mit `get()` wird das nächste Token geliefert und der Stream weiter geschaltet.
   - Mit `peek()` wir das nächste Token geliefert und der Stream **nicht** weiter geschaltet, sodass geprüft werden kann welches Token kommen würde.

Anmerkungen:

1. Geben Sie für Ihre Problemlösungen auch Lösungsideen an.
2. Kommentieren Sie Ihre Algorithmen ausführlich.
3. Strukturieren Sie Ihre Programme sauber.
4. Geben Sie ausreichend Testfälle ab und prüfen Sie alle Eingabedaten auf ihre Gültigkeit.