Übung 8

Abgabe: 19. Juni 2023, 18:00 Uhr

Aufgabe 8.1: Haeufigkeiten [12 Punkte, Link zur Abgabe] In dieser Aufgabe entwickeln wir ein Programm, das zählt, wie oft welches Wort in einem Text vorkommt. Dazu stecken wir verschiedene Datenstrukturen und Algorithmen zusammen.

Listing 1: Beispielein- und Ausgabe

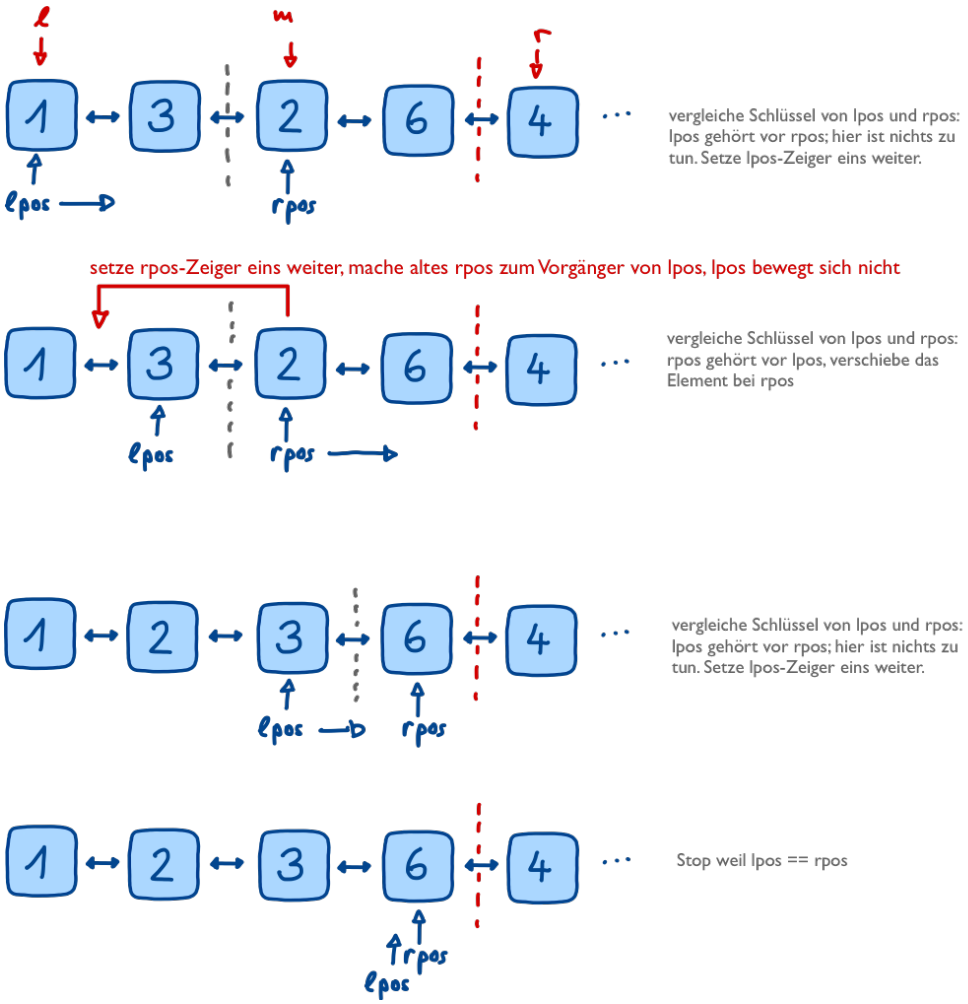
```
> cat example.in
Wir programmieren
Falsche Ausgabe warum
Programmieren wir?
> ./wordcount < example.in
ausgabe 1
falsche 1
programmieren 2
warum 1
wir 2
```

Das Programm soll für jedes Wort w im Eingabetext das Wort w und die Anzahl von Vorkommen von w ausgeben. Wir unterscheiden nicht zwischen Groß- und Kleinschreibung des Wortes; alle Wörter werden in Kleinbuchstaben ausgegeben. Die Wörter sollen in alphabetischer Reihenfolge ausgegeben werden.

- Sie finden im Repository eine HashMap, die Null-Terminated-Strings auf ganze Zahlen abbildet. Verwenden Sie die HashMap, um die Häufigkeiten der Wörter zu zählen.
- Die HashMap benötigt eine Funktion `str2int`, die einen String in eine ganze Zahl umwandelt, die anschließend mit einer Hashfunktion gehasht werden kann (s. Kapitel 5.2.3/Listing 5.5 im Aldat-Skript). Da wir in C 8-Bit-Chars verwenden, setzen wir $R = 2^8 = 256$ anstatt $R = 2^{16}$ wie im Skript. Das ist die einzige Änderung, die in `HashMap.c` nötig ist.
- *Tipp:* Fügen Sie außerdem jedes Wort *einmal* in eine Arrayliste ein (auch im Repository gegeben). Sie finden im Repository eine Implementierung von MergeSort.

Folgen Sie den Hinweisen in `main.c`!

Aufgabe 8.2: sortierte-listen [8 Punkte, Link zur Abgabe] Wir haben in der Vorlesung haben wir die MergeSort-Variante *Bottom-Up*-MergeSort gesehen, die im Gegensatz zum konventionellen (Top-Down)-MergeSort auch verkettete Listen effizient sortieren kann – und zwar in-place, d.h. ohne zusätzlichen Speicherplatz!. Im Repository finden Sie eine entsprechende Implementierung; es fehlt aber die `merge`-Methode. Ein Beispiel für einen Methodenaufruf:



Es treten zwei Fälle auf:

- Ist der Schlüssel des Elements bei `l_pos` kleiner als der des Elements bei `r_pos`, so bewegt sich der Zeiger `l_pos` in der Liste eine Position weiter.
- Anderenfalls gehört das Element bei `r_pos` in der sortierten Liste vor das Element bei `l_pos`. Wir merken uns, auf welches Element `e` der Zeiger `r_pos` gerade zeigt und bewegen ihn dan eine Position weiter in der Liste. Dann verschieben wir `e` so, dass es in der Liste direkt vor dem Element steht, auf das `l_pos` zeigt.

Wir beobachten, dass `r_pos` immer auf den Beginn des rechten Teilarrays zeigt, so dass es zwei

Stopp-Bedingungen für die Methode gibt: Ist `l_pos >= r_pos` so wurde das linke Teilarray komplett gemerged. Ist `r_pos >= r`, so wurde das rechte Teilarray komplett gemerged.

Da die doppelt verkettete Liste ein `tail`-Element besitzt, können wir davon ausgehen, dass `r` immer auf ein Listenelement zeigt.

Ihre Aufgabe: Implentieren Sie die Methode `merge` und eine Methode `dlist_move`, die einer Liste ein Element verchiebt. Lassen Sie den restlichen Code unverändert.

Bemerkung: Das Ergebnis ist ein Sortierverfahren mit Worst-Case-Laufzeit $\Theta(n \log n)$, das mit konstantem zusätzlichem Speicher und ohne Rekursion auskommt!

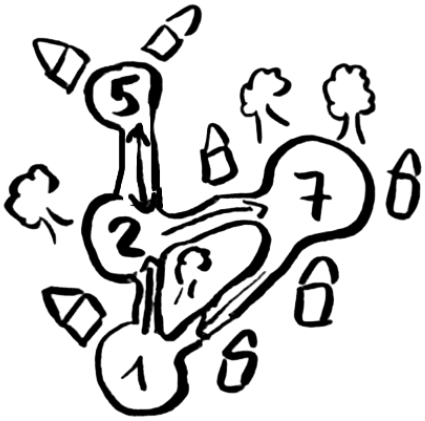
Aufgabe 8.3: Eiswagen [10 Punkte, Link zur Abgabe] Ein Eiswagen darf an verschiedenen Straßenkreuzungen stehen, jede dieser Straßenkreuzungen hat eine eindeutige ID (`int`). Lesen Sie von der Standardeingabe zunächst ein, wie viele Straßenkreuzungen es gibt. Anschließend lesen Sie die IDs der Kreuzungen ein. Die Kreuzungen sollen in einer Arrayliste gespeichert werden.

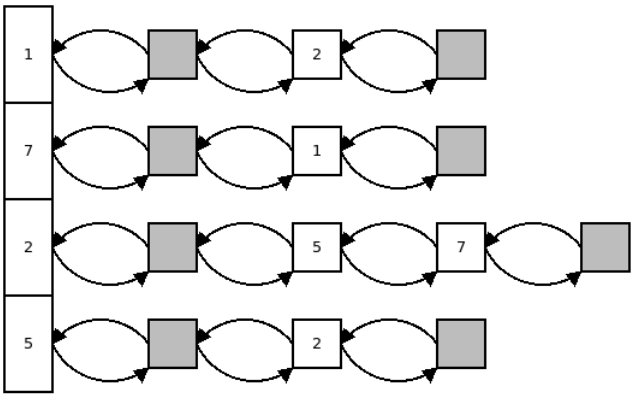
Neben der ID der Kreuzungen verwaltet die Arrayliste noch einen Verweis auf eine doppelt verkettete Liste von denjenigen Straßen, die von der Kreuzung ausgehen. Genauer speichert die Liste an Kreuzung i die IDs der Kreuzungen, die zu i benachbart sind. Die Liste soll aufsteigend sortiert gespeichert werden.

Auf der Standardeingabe folgen nach den Kreuzungen Straßeneinträge. Die Einträge haben eines von zwei Formaten: Im ersten Format werden Straßen hinzugefügt (Beispiel: `a 2 3` – fügt Straße von 2 nach 3 hinzu). Das zweite Format entfernt Straßen (Beispiel: `r 2 3` – entfernt Straße von 2 nach 3), z.B. wegen Baustellen. Straßen haben selber keine ID sondern werden angegeben, indem die ID der Start- und Zielkreuzung angegeben werden.

Achten Sie auf eine saubere dynamische Speicherverwaltung für die Straßen.

Sollten Sie Schwierigkeiten haben, sich die Datenstruktur vorzustellen, betrachten Sie folgende zwei Abildungen. Links ein „Straßenplan“ und rechts die Darstellung der Datenstruktur.





Aufgabe 7.3: Projektmanagement [10 Punkte, Link zur Abgabe] Sie dürfen auch diese Woche noch Aufgabe 7.3 vom letzten Übungsblatt bearbeiten. Beachten Sie bitte, dass sich die Testfälle gegenüber der ursprünglichen Version des Übungsblattes geändert haben!

Wenn Sie die ursprüngliche Version der Aufgabe bearbeitet haben, kontrollieren Sie bitte Ihre Lösung.