Übung 7

Abgabe: 12. Juni 2023, 18:00 Uhr

Aufgabe 7.1: Maximum Subarray [10 Punkte, Link zur Abgabe] Das Maximum-Subarray-Problem fragt nach einem zusammenhängenden Teilarray $A[l, \ldots, r]$ eines Arrays $A[0, \ldots, n-1]$ von ganzen Zahlen, das die Summe $A[l] + \cdots + A[r]$ seiner Elemente maximiert. Es ist $l \leq r$, so dass $A[l, \ldots, r]$ immer mindestens ein Element enthält. Zum Beispiel besitzt das Array A

Index	0	1	2	3	4	5	6	7	8
A	-10	9	-2	1	-5	4	3	-5	1

das Teilarray $A[1, ..., 6]$ mit Summe 10, aber kein Teilarray mit einer Summe > 10. Daher ist $A[1, ..., 6]$ ein Maximum-Subarray von A.

Algorithmus A berechnet für jedes Paar $l, r \in \{0, \ldots, n-1\}$ mit $l \leq r$ die Summe $A[l] + \cdots + A[r]$. Dabei speichert er die größte Summe, die auftritt und gibt sie am Ende aus.

Algorithmus B speichert für $k = 0, \ldots, n-1$ zwei Werte: In M_k berechnen wir die Summe eines Maximum-Subarrays von $A[0, \ldots, k]$ und geben M_{n-1} aus. In R_k berechnen wir die größte Summe eines *rechtsbündigen* Subarrays von $A[0, \ldots, k]$; das ist eines, das als rechte Grenze k hat.

Zur Berechnung von M_k und R_k setzen wir $M_0 = R_0 = A[0]$. Anschließend erhalten wir R_{k+1} als $\max\{A[k+1], A[k+1] + R_k\}$: Das maximale rechtsbündige Teilarray von $A[0, ..., k+1]$ enthält in jedem Fall das Element $A[k+1]$. Wenn es sich lohnt, das Teilarray nach links zu erweitern hat es die Summe $R_k + A[k+1]$, ansonsten enhält es nur $A[k+1]$.

Außerdem erhalten wir M_{k+1} als $\max\{M_k, R_{k+1}\}$: Entweder endet das maximum Subarray von $A[0, \ldots, k+1]$ an Position $k+1$ (dann ist es in R_{k+1} berücksichtigt) oder es endet an einer Position $< k+1$ und ist in M_k berücksichtigt.

Algorithmus C berechnet für alle $l = 0, \ldots, n-1$ und alle $r = 0, \ldots, n-1$ die Summe A^l_r der Elemente $A[l], \ldots, A[r]$. Dabei ergibt sich A^l_r als $A^l_{r-1} + A[r]$. Am Ende geben wir das Maximum über alle A^l_r aus. *Es ist nicht notwendig, alle A^l_r zu speichern.*

Wählen Sie einen der drei Algorithmen aus und implementieren Sie ihn in `max-subarray.c`*. Notieren Sie Ihre Wahl in einem Kommentar. Nur der schnellste der drei Algorithmen besteht alle Testfälle innerhalb des Zeitlimits!*

Beantworten Sie für drei Punkte außerdem die Ankreuzfragen im Repository.

Aufgabe 7.2: Sortieralgorithmen [10 Punkte, Link zur Abgabe] Im Repository dieser Aufgabe finden Sie einige fertig implementierte Sortieralgorithmen. Ihre Aufgabe: Schreiben Sie passende Testfälle!

Welche Anforderungen die Testfälle erfüllen sollen, erfahren Sie im Repository in der Datei `testfaelle.md`. Schreiben Sie Ihre Lösung in die Dateien im Verzeichnis `testcases`.

Sie können die Implementierungen lokal testen, indem Sie mit `make` kompilieren und dann z.B. `./sortme i < example.in` für InsertionSort mit der Eingabe `example.in` aufrufen. Weitere Wahlmöglichkeiten sind `m` (Bottom-Up-Mergesort), `q` (QuickSort) und `s` (SelectionSort). Das Programm erzeugt keine Ausgabe.

Sie finden im Repository ein weiteres Programm `randnums.c`, das Zufallszahlen erzeugen kann.

Listing 1: Beispielworkflow zum Erstellen und Testen einer Eingabe

```
1 > make
2 > ./randnums 20000 > random-input.in
3 > time ./sortme s < random-input.in
4 0.17s user 0.00s system 93\% cpu 0.186 total
```

Um die geforderten Testfälle zu erzeugen, schreiben Sie sich weitere Programme `generator1.c` und `generator2.c`. Laden Sie die beiden Generatoren, aber auch die von Ihnen erzeugen Eingaben im Repository hoch!

Die Laufzeiten auf Ihrem Computer werden in der Regel nicht mit den gemessenen Laufzeiten auf dem Server übereinstimmen. Als Hilfestellung zeigt `eval.py` einen (geschätzten) Geschwindigkeitsfaktor an. Ein Faktor von x bedeutet, dass die Laufzeiten auf ihrem Computer ca. um den Faktor x von den Serverlaufzeiten abweichen.

Hinweis: Viele CPUs können den maximalen Takt ohne ausreichende Kühlung nur für sehr kurze Zeit halten. Insbesondere Laptops stellen in der Regel keine ausreichende Kühlung zur Verfügung. Es kann daher passieren, dass die Laufzeiten – vorallem bei wiederholten Tests – lokal auf Ihrem Computer stark variieren.

Verändern Sie die vorgegebene Implementierung nicht.

Aufgabe 7.3: Projektmanagement [10 Punkte, Link zur Abgabe] **Für diese Aufgabe haben Sie eine Woche länger Zeit (bis zum 19.06.2023, 18 Uhr).**

In dieser Aufgabe simulieren wir eine Softwarefirma – Ähnlichkeiten mit realer Softwareentwicklung sind beabsichtigt, aber unwahrscheinlich ;)

Die Firma entwickelt in einer Pipeline mit drei Teams: Wird ein neues Projekt akquiriert, kommt es zunächst zum Planungsteam. Dieses gibt das Projekt an das Entwicklungsteam weiter, das es schließlich der Qualitätssicherung übergibt. Jedes Projekt hat eine `id` und einen Wert `value`.

Zur Gewinnmaximierung arbeitet jedes Team immer an dem Projekt, das aktuell in der Abteilung des Teams liegt und den größten Wert für die Firma darstellt, aber noch von keinem anderen Team bearbeitet wird. Einmal begonnen Arbeit an einem Projekt wird aber nicht mehr unterbrochen.

Schreiben Sie ein Programm, dass die Abläufe simuliert, indem Sie für jedes der drei Teams einen Max-Heap (die Warteschlange mit unbearbeiteten Projekten des Teams) und ein aktuelles Projekt verwalten. Die Eingabedatei gibt an, wann neue Projekte akquiriert werden und wann eine Abteilung mit einem Projekt fertig wird (s. Beispiel auf der nächsten Seite).

Sie finden im Repository die Implementierung eines MaxHeaps.

- Passen Sie diese Implementierung so an, dass der MaxHeap Pointer auf die `Project`-Struct aus `project.h` verwalten kann. Tipp: Schauen Sie zuerst in `binheap.h`.
- Verändern Sie den Heap so, dass er das `value`-Attribut der Struct als Priorität verwendet.
- Schreiben Sie Ihre Simulation in die Datei `main.c`. Die passenden Meldungen für die Ausgabe sind bereits vordefiniert; verwenden Sie die entsprechenden Makros!
- Alle Projekte sollen mit `malloc` allokiert werden. Achten Sie darauf, den Speicher wieder freizugeben!
- Die Vorgehensweise beim wenn ein Projekt beendet wird ist so: Zuerst wird das Projekt beendet, dann wird es eingereiht und dann erst beginnt das nächste Team mit der Arbeit (sofern es gerade kein aktives Projekt hat).
- Beendetete Projekte werden immer in der Warteschlange des nächsten Teams eingereiht, auch wenn das nächste Team gerade kein aktives Projekt hat.

Listing 2: Beispieldurchlauf

```
> cat example.in
p 1234 1000
p 1235 1500
f 0
p 2035 2000
f 0
f 1
f 0
f 1
> ./pm < example.in
Projekt 1234 wird bei Team 0 eingereiht.
Team 0 beginnt 1234.
Projekt 1235 wird bei Team 0 eingereiht.
Team 0 beendet 1234.
Projekt 1234 wird bei Team 1 eingereiht.
Team 0 beginnt 1235.
Team 1 beginnt 1234.
Projekt 2035 wird bei Team 0 eingereiht.
Team 0 beendet 1235.
Projekt 1235 wird bei Team 1 eingereiht.
Team 0 beginnt 2035.
Team 1 beendet 1234.
Projekt 1234 wird bei Team 2 eingereiht.
Team 1 beginnt 1235.
Team 2 beginnt 1234.
Team 0 beendet 2035.
Projekt 2035 wird bei Team 1 eingereiht.
Team 1 beendet 1235.
Projekt 1235 wird bei Team 2 eingereiht.
```