Technische Universität München
Fakultät für Informatik, I-14
Prof. Harald Räcke
Robin Münk

SoSe 2025
Übungsblatt 7
6. Juni 2025

Algorithmen und Datenstrukturen

Aufgabe 1 Merge Sort und Quick Sort

In der Vorlesung haben Sie mittlerweile zwei weitere Sortieralgorithmen kennengelernt, *Merge Sort* und *Quick Sort*, die beide nach dem *Divide-and-Conquer*-Prinzip funktionieren. Vergegenwärtigen Sie sich nochmals deren Ablauf! Zur Verdeutlichung des Ablaufs der Algorithmen können Sie sich folgende Videos anschauen: Merge Sort, Quick Sort

Die Idee bei *Divide-and-Conquer* ist stets, das eigentliche Problem in kleinere Teilprobleme gleicher Art aufzuspalten, und diese dann separat zu lösen. Aus den Ergebnissen der Teile lässt sich dann einfach die Gesamtlösung erzeugen. Wichtig ist, dass dann auch die Teilprobleme entweder trivial oder ihrerseits aufspaltbar sind, so dass der *Divide-and-Conquer*-Ansatz rekursiv arbeitet.

a) Gegeben sei das Array $A = \{d, b, f, g, e, a, c\}$.

Sortieren sie das Array A lexikographisch mittels *Merge Sort*. Wählen Sie für ungerade n folgende Strategie:

$$A = \{a_1, \ldots, a_n\} \rightarrow A_{\text{left}} = \{a_1, \ldots, a_{\lfloor \frac{n+1}{2} \rfloor}\},\ A_{\text{right}} = \{a_{\lfloor \frac{n+1}{2} \rfloor + 1}, \ldots, a_n\}$$

Zeichnen Sie die Zwischenschritte der Sortierung auf, sodass die Aufteilungen und Rekombinationen ersichtlich werden.

b) Gegeben sei das Array $A = \{2, 9, 5, 4, 8, 3, 1, 2\}$.

Sortieren sie das Array A mit Quick Sort wobei Sie als Pivot-Element stets das letzte Element wählen.
Zeichnen Sie die Zwischenschritte der Sortierung auf, sodass die Aufteilungen und Rekombinationen ersichtlich werden.

Aufgabe 2 Problem des Josephus

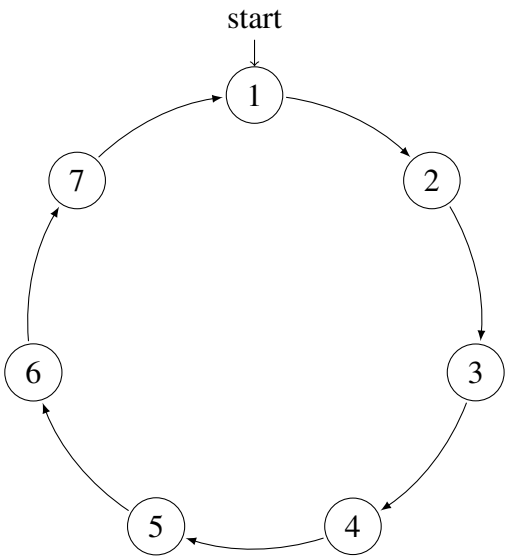
Hintergrund:

> *Das Problem wurde nach dem jüdischen Historiker Flavius Josephus benannt, welcher sich 67 n. Chr. beim Kampf um die galiläische Stadt Jotapata mit 40 weiteren Männern in einer Höhle vor den Römern versteckt hielt. Als das Versteck verraten wurde, sicherten die Römer Josephus freies Geleit zu für den Fall, dass er das Versteck verließ. Seine Gefolgsleute drohten allerdings ihn umzubringen und wollten lieber sterben, als den Römern in die Hände zu fallen. Daraufhin machte Josephus den Vorschlag eines kollektiven Suizids, in dem sich alle im Kreis aufstellen und jeder seinen linken Nachbarn töten sollte. Er stellte sich an die 19. Stelle, blieb damit als Letzter übrig, ergab sich den Römern und überlebte.*
> Quelle: Wikipedia

Welcher Platz bleibt als letzter übrig, bei n Anwesenden? Wir werden in dieser Aufgabe eine allgemeine Version dieses Problems mithilfe einer *Cyclic Linked List* lösen.

Problemdefinition: Seien n Elemente in einem Kreis angeordnet und durchnummeriert (beginnend mit 1). Beginnend mit dem k-tem Element wird jedes k-te Element gelöscht bis nur noch ein Element übrig bleibt. Der Index dieses Elements ist die Lösung. Das Originalproblem entspricht dem Fall $k = 2$.

Beispiel für $n = 7, k = 3$: Ursprüngliche Liste: $[1,2,3,4,5,6,7]$. Folgende Elemente werden in der gegebenen Reihenfolge gelöscht: $3,6,2,7,5,1$. Die Antwort ist: 4.



Eine *Cyclic Linked List* funktioniert genau wie eine normale Single Linked List, insbesondere ist die *ListNode*-Klasse identisch (ein Integer als Datenfeld und ein Pointer auf den nächsten Knoten). Der einzige Unterschied ist, dass das letzte Element in der Liste nicht *NULL* als Nachfolger hat, sondern wieder auf das erste Element in der Liste zeigt.

Implementieren Sie eine Funktion `josephus(int n, int k)`, die das Problem des Josephus mithilfe einer *Cyclic Linked List* mit Elementen für ein gegebenes $n \geq 2$ und $2 \leq k \leq n$ simuliert. Die Funktion gibt sowohl die Reihenfolge, in der die Element aus der Liste gelöscht werden, als auch das Label des letzten Überlebenden in einem Paar aus `vector<int>` und `int` zurück.

Sie können für diese Aufgabe das Testing Framework verwenden. Die Klasse `CyclicLinkedList` ist gegeben. Achten Sie darauf, keine Memory Leaks zu hinterlassen, also den Speicherplatz von ggf. gelöschten ListNodes wieder freizugeben. Der Destruktor der Klasse `CyclicLinkedList` iteriert durch die Liste und gibt den verwendeten Speicherplatz wieder frei.

Aufgabe 3 (P) Linked List - Polynome

Sei ein Polynom der Form

$$p(x) = c_n x^n + c_{n-1} x^{n-1} + \cdots + c_1 x + c_0, \quad n \geq 0$$

gegeben. Eine mögliche Datenstruktur zur Darstellung von Polynomen ist eine einfach verkettete Liste. Ein Listenelement enthält dabei den Koeffizienten c_i sowie den Exponenten i. Die einzelnen Listenelemente sind dabei nach absteigenden Exponenten geordnet. Listenelemente mit dem Koeffizienten 0 kommen nicht vor.

Sie können für diese Aufgabe das Testing Framework verwenden.

```
class Polynomial {
    public:
    ListNode *head;
    void print();
}
```

```
struct ListNode {
    int i;      // Exponent
    float ci;   // Koeffizient
    ListNode *next;
}
```

Implementieren Sie folgende Methoden für die Klasse `Polynomial`:

a) `float evaluate(float x)`: Die Methode bekommt als Eingabe einen Wert x und wertet das Polynom an der Stelle x aus.

b) `bool equalTo(Polynomial& other)`: Die Methode bekommt als Eingabe ein zweites Polynom `other` und gibt aus, ob die Polynome gleich sind.

Hinweise:

- Sie können die Methode `std::pow(a, b)` verwenden, um a^b zu berechnen.
- Vielleicht helfen Ihnen sinvolle rekursive Hilfsfunktionen. Arbeiten Sie systematisch die möglichen Fälle ab (beide Polynome zu Ende, nur eins zu Ende, keins zu Ende, ...).
- In der Klasse `Polynomial` gibt es eine print Funktion zum debuggen.
- Als Hilfestellung können Sie sich an den Hinweisen in der Datei `blatt7hint.cpp` orientieren.