Aufgaben

Multiplikation

Multipliziere 8430 und 7763 mit Karazuba-Offman.

Binäre Heaps

```
A1 := [13, 12, 1, 14, 7, 5, 0, 6, 9, 8, 11, 2, 10, 4, 3]
A2 := [1, 8, 15, 14, 0, 0, 2, 10, 9, 7, 12, 11, 15, 2, 9]
H1 := [0, 2, 1, 3, 7, 9, 4, 12, 6, 8, 13, 10, 11, 5, 14]
H2 := [0, 0, 1, 3, 2, 9, 5, 12, 11, 5, 2, 15, 14, 13, 10]
```

1) Erzeuge mit `make_heap()` aus `A1` und `A2` gültige min-Heaps
2) `H1` und `H2` sind min-heaps. Führe `delete_min` auf ihnen aus
3) Füge 16 in `H1` und 6 in `H2` ein

Hashmaps

1) Was sind die zwei grundlegenden Arten Hashmaps zu implementieren?
2) Diskutiere Vor- und Nachteile der beiden Methoden

Sortieren – Basics

1) Welche Sortieralgorithmen kennt ihr aus der Vorlesung?
2) Welche Best-, Average- und Worst-case-Laufzeiten haben diese?
3) Welche Vorteile, Nachteile und Beschränkungen existieren für jeden der Algorithmen?
4) Verwende Bucket-sort um das folgende Array zu sortieren:
   `[8, 4, 3, 10, 7, 7, 6, 3, 2, 6]`
5) Verwende LSD-Radixsort um das folgende Array zu sortieren:
   `[939, 73, 808, 396, 302, 895, 403, 730, 797, 136]`

Sortieren – Anwendung

1) Welchen Sortieralgorithmus würdest du in jeder der folgenden Situationen verwenden und warum?

    1) Du hast ein sehr kleines Array von Integern
    2) Du sollst ein Array ohne dynamische Speicherallokation sortieren; versuchte DoS-Angriffe durch bösartig gewählte Inputs sind wahrscheinlich

3) Du hast ein Array von sehr großen, in-place gespeicherten Elementen und sehr billigen Vergleichen

2) Du hast eine Menge fester Größe bereits sortierter Arrays. Wie erzeugst du ein sortiertes Arrays das die Vereinigung aller Elemente dieser Arrays enthält?

Quickselect

Bestimme mit Quickselect das dritt- und das siebt-größte Element der folgenden Arrays:

```
A1 := [6, 9, 11, 7, 1, 8, 4, 0, 5, 14, 3, 10, 12, 2, 13]
A2 := [14, 0, 13, 11, 4, 5, 10, 12, 3, 2, 7, 8, 6, 1, 9]
A3 := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

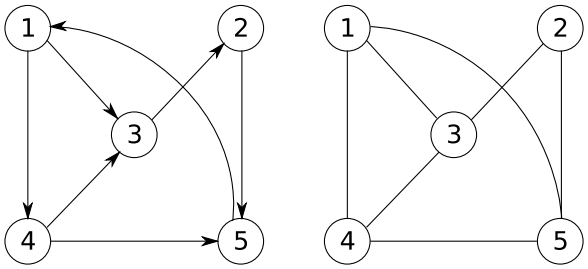
- Verwende jeden der beiden vorgestellten Partitionierungsalgorithmen mindestens für ein Array.
- Wähle das erste/letzte/mittlere Element des betrachteten Teilarrays als Pivot.
- Was sind die Laufzeit-Eigenschaften von Quickselect?

Master-Theorem

1) Löse die folgenden Rekurrenzen:
   1) $f(n) = 3n + 2 \cdot f\left(\frac{n}{5}\right)$
   2) $f(n) = \pi n + 3 \cdot f\left(\frac{n}{3}\right)$
   3) $f(n) = 7n + 4 \cdot f\left(\frac{n}{2}\right)$
2) Wie lautet das Master-Theorem?

Breitensuche

Gegeben seien die folgenden Graphen:



Führe Breitensuchen ausgehend von den Knoten 1 und 2 aus.

Graphdatenstrukturen

1) Schreibe die Graphen der vorherigen Aufgabe als
    1) Adjazenzmatrix
    2) Adjazenzliste
    3) Adjazenzfeld
2) Zeichne die folgenden Graphen:
    1) Adjazenzarray:

```
V = [0, 0, 1, 4, 6, 7]
E = [0, 2, 0, 1, 0, 3, 4]
```

    2) Adjazenzliste:

```
[[], [0, 2], [1, 3], [0, 1, 2]]
```

    3) Adjazenzmatrix: $\begin{pmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix}$

Suchbäume

```
A1 := [9, 0, 1, 14, 12, 8, 10, 13, 5, 2, 11, 4, 7, 3, 6]
A2 := [9, 1, 7, 14, 10, 6, 10, 5, 12, 0, 13, 14, 2, 4, 3]
A3 := [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]
```

1) Erzeuge aus den obigen Arrays unbalancierte Binärbäume, indem du ihre Elemente der Reihe nach einfügst.
2) Beschreibe wie a,b-Bäume prinzipiell funktionieren
3) Diskutiere Vor- und Nachteile von sortierten Listen mit a,b-Bäumen als Navigationsdatenstruktur im Vergleich zu anderen Containerdatenstrukturen

Rotieren

Beschreibe einen Algorithmus, der einen Teil eines Containers an eine andere stelle verschiebt:

```
fun rotate(a: Array, old_start: index,
           old_end: index, new_start:index): index
```

Der Algorithmus soll:

- höchstens Linearzeit benötigen
- mit konstant viel zusätzlichem Speicher auskommen.
- zurückgeben, wohin (Index) verschoben wurde.

Tipp: Löse diese Aufgabe vor der nächsten.

Stabile Partitionierung

Schreibe einen Algorithmus der ein Array bezüglich einer Eigenschaft stabil partitioniert:

```
fun stable_partition(a: Array, f:Predicate) -> index
```

Der Algorithmus soll:

- den Index des ersten Elemens zurückgeben, dass die Eigenschaft nicht erfüllt (oder `a.size()` falls es kein solches Element gibt).
- maximal logarithmisch viel zusätzlichen Speicher benötigen.
- eine Laufzeit in $O(n \log n)$ haben.

Welche Laufzeit wäre möglich, wenn linear viel zusätzlicher Speicher verfügbar wäre und wie würde dies funktionieren?

Feedback

Hier ist Platz für Anregungen, Kritik, Wünsche, ...