Übung 12: Priority Queue, Substring Search

Aufgabe 1: Implementierung des ADT Priority Queue über einen MinHeap

Ein `Min-Heap` wird als Array implementiert. In dem betreffenden Array seien die Elemente in der folgenden Reihenfolge gespeichert: $\langle 5, 9, 8, 13, 14, 10, 16, 20, 17 \rangle$. Alle Operationen der Priority Queue seien genauso wie im Pseudocode der Vorlesung implementiert.

a) Zeichnen Sie die Repräsentation als Baum und überprüfen Sie ob die Heap-Eigenschaft für einen `Min-Heap` erfüllt ist. Stellen Sie ggfs. den Heap durch geeignete Operationen wieder her.

b) Zeichnen Sie den Heap nach dem Ausführen der Operation `EXTRACT-MIN()`!

c) Führen Sie nun die Operation `INSERT(7)` auf dem Ergebnis von Aufgabe b) aus. Zeichnen Sie das Ergebnis in der „Baumdarstellung“.

d) Welche Worst-Case Komplexität (O-Notation) haben die Operationen `MINIMUM()`, `EXTRACT-MIN()` und `INSERT(k)` in Abhängigkeit der Eingabegröße *n?*

e) Wie wäre die Komplexität der Operationen aus d), falls Sie eine Priority Queue nicht durch einen `MinHeap`, sondern durch einen Red-Black Tree implementieren?

Aufgabe 2: Boyer-Moore

a) Skizzieren Sie die Belegung des `right[]` Array, das der Boyer-Moore Algorithmus für das folgende Muster berechnet! Grundlage ist der Java-Code der Vorlesung mit $R = 256$. Beachten Sie die folgende Zeichentabelle: http://www.asciitable.com/
Muster: `A B R A C A D A B R A`

b) Wie viele Vergleiche von Textzeichen benötigt der Boyer-Moore Algorithmus der Vorlesung in folgendem konkretem Beispiel?
Text: `G C A A T G C C T A T G G G C T A T G T G`
Muster: `T A T G T G`

c) Geben Sie ein Beispiel an, bei dem die Laufzeit des Boyer-Moore Algorithmus der Vorlesung sehr ungünstig ist!

Aufgabe 3: Rabin-Karp

a) Was ist der Unterschied zwischen einem Monte-Carlo und einem Las-Vegas Algorithmus? Was unterscheidet einen Las-Vegas Algorithmus von einem „normalen Algorithmus“?

b) Verwenden Sie zur Suche den Rabin-Karp Algorithmus der Vorlesung auf dem folgenden Beispiel. Wie viele falsche Treffer liefert der Algorithmus unter den folgenden Annahmen?
- Text: 3 1 4 1 5 9 2 6 5 3 5 8 9 7 9 3
- Muster: 2 6
- Hashfunktion: $q = 11$ (rechnet „modulo 11“)
- Basis: $R = 10$, der String besteht also nur aus Ziffern.
- Es wird die Monte-Carlo Variante verwendet.

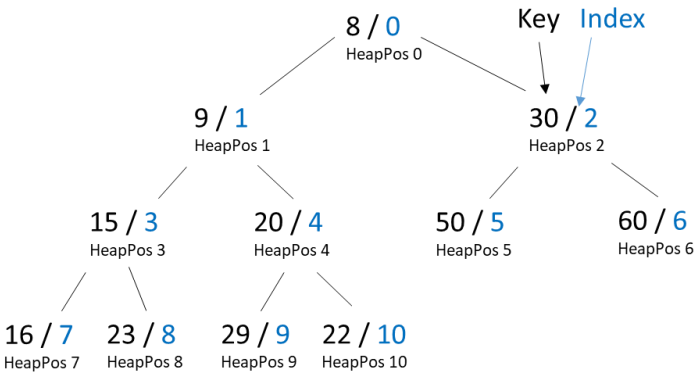
c) Beschreiben Sie in 1-2 Sätzen, wie Sie den Algorithmus von Rabin-Karp erweitern würden, so dass Sie herausfinden können, ob ein beliebiges Muster aus einer Menge vorgegebener Muster (alle gleicher Länge) im Text enthalten ist.

Aufgabe 4: Implementierung einer indizierten Priority Queue

Setzt man eine Priority Queue ein, so ist es oft wichtig, dass man als Anwender schnell zu bestimmten Einträgen der Priority Queue navigieren kann. Beim Dijkstra-Algorithmus muss man z.B. schnell an die Position des Heap-Arrays navigieren, an dem ein bestimmter Schlüssel/Knoten steht, um dessen Wert zu verringern: `decreaseKey(int i, Key key).`

Deshalb wird jeder Schlüssel mit einem festen ***Index / Handle*** `i` assoziiert. Diesen Index legt bereits der Anwender beim Einfügen in die Queue fest: `insert(int i, Key key).` Danach gehören ein Key[1] und sein Index untrennbar zusammen, der Key wird immer an `keys[index]` gespeichert bleiben. Mit dem Index kann der Anwender später direkt zum Eintrag im *MinHeap* navigieren. Das eigentliche Heaparray ist das Array `pq`, das statt des eigentliches Keys aber den Index speichert.

Verständnisfrage: Versuchen Sie die Darstellung auf Folie 34 der Vorlesung zu verstehen. Was ändert sich hier, wenn man die 9 (Index 1) löscht? (wird in Übung besprochen)



pq	0	1	2	3	4	5	6	7	8	9	10

heapPos	0	1	2	3	4	5	6	7	8	9	10
keys	8	9	30	15	20	50	60	16	23	29	22
Index	0	1	2	3	4	5	6	7	8	9	10

Ergänzen Sie die vorgegebene Klasse `PriorityQueue` wie folgt. Es ist nicht viel zu tun.

- `public void delete(int i)`: Löscht den Schlüssel mit Index `i` aus der Queue. Nehmen Sie an, dass zunächst das Element ganz rechts an dessen Stelle rückt (wie bei Heapsort).
- `public void changeKey(int i, Key key)`: Ändert Wert des Schlüssels mit Index `i` auf `key`.

Verwenden Sie soweit als möglich bereits vorhandene Methoden der Klasse. Vor allem `heapify(.)` und `swim(.)` sind hilfreich. Die JUnit-Testklasse erzeugt initial den oben abgebildeten Heap.

[1] Könnte eine beliebige Klasse sein, die Comparable implementiert, z.B. String