Algorithmen und Datenstrukturen
Sommersemester 2022
Woche 3

Kevin Angele, Tobias Dick, Oskar Neuhuber,
Andrea Portscher, Monika Steidl, Laurin Wischounig

Abgabe bis 29.03.2022 23:59
Besprechung im PS am 31.03.2022

Aufgabe 1 (2 Punkte): Stapel

Palindrome sind Wörter oder Sätze, die von vorne und hinten gelesen das gleiche Wort ergeben.

1. Wie kann man mit Hilfe eines Stapels erkennen, ob das eingegebene Wort ein Palindrom ist (Satzzeichen und Leerzeichen ignorierend)? Zeichnen Sie dafür die Überlegungen auf und implementieren Sie einen Algorithmus.
2. Teste folgende Wörter und Sätze. Welche sind Palindrome?
   (a) Abba
   (b) Rentner
   (c) Angela
   (d) Lagerregal
   (e) Algorithmen und Datenstrukturen
   (f) Saippuakivikauppias (Finnisch für Specksteinhändler)
   (g) Die Liebe ist Sieger, stets rege ist sie bei Leid.
   (h) Ist das die Loesung?
   (i) Geist ziert Leben, Mut hegt Siege, Beileid tragt belegbare Reue, Neid dient nie, nun eint Neid die Neuerer, abgelebt gart die Liebe, Geist geht, umnebelt reizt Sieg.
3. Welche Vorteile und Nachteile hat diese Implementierung? Welche Komplexität weisen die von Ihnen verwendeten Funktionen auf? Was ist die Laufzeitkomplexität Ihres Algorithmus?

Lösung:
Mögliche Implementierung: `Palindrome.java`

Vorteile: Regex erlaubt einfache Zerlegung der Sätze nach bestimmten regeln, ArrayList passt sich automatisch der Anzahl an Elementen an

Nachteile: Einfügen von Elementen wenn Kapazität erreicht ist ist $O(n)$, da die Elemente in das neue Array kopiert werden müssen

Laufzeitkomplexität: `push`: $O(1)$ (Annahme: Arrays werden nicht voll); `pop`: $O(1)$ (Index des zu löschenden Elements ist bekannt); `isEmpty`: $O(1)$; `top`: $O(1)$;

Aufgabe 2 (2 Punkte): Warteschlange

Verwenden Sie den zur Verfügung gestellten Java-Code `QueueBenchmark.java` einer Warteschlange mit Hilfe einer verketteten Liste, einer Double-Ended Queue und einer Double-Ended Queue mit maximaler Größe.

1. Bestimme welche Implementierung am Schnellsten ist - warum?
2. Was sind die Vor- und Nachteile einer verketteten Liste, einer Double-Ended Queue und einer Double-Ended Queue mit maximaler Größe?

Lösung:

1. Aufzählung vom Schnellsten zum Langsamsten: ArrayDeque<>(MAX_SIZE), ArrayDeque<>(), LinkedList()
2. Vor- und Nachteile
   - LinkedList[1]: wenn Element hinzugefügt wird, wird ein neuer Knoten erstellt; flexibler (e.g. NULL Elemente), $O(1)$ beim Einfügen/Löschen von Elementen am Anfang
   - ArrayDeque[2]: hinzufügen und löschen an beiden Enden (falls das Array vergrößert werden muss, wird das Array verdoppelt und die Elemente kopiert $O(n)$)
   - ArrayDeque mit mazimaler Größe: muss nicht resized werden; aber größerer Speicherverbrauch

Aufgabe 3 (2 Punkte): Abstrakte Datentypen

1. Wähle einen am Besten geeigneten abstrakten Datentyp aus, um folgendes Problem zu lösen. Skizzieren Sie hierzu informell die Anwendung der Methoden des gewählten abstrakten Datentyps:
   Es geht um die Planung von Tasks für mehrere Prozessoren, wobei die Threads für jeden Prozessor separat verwaltet werden. Ein Prozessor führt immer den ersten vorhandenen Thread aus. Wenn sich der aktuelle Thread teilt, wird dieser wieder ganz vorne eingefügt und ein neuer Thread wird ausgeführt. Wenn einer der Prozessoren die Ausführung seiner eigenen Threads beendet hat (es steht kein weiterer Thread für diesen Prozessor zur Verfügung), dann kann dieser einen Thread von einem anderen Prozessor *stehlen*. Hierbei wird der letzte Thread eines anderen Prozessors entnommen und ausgeführt.
2. Identifiziere die momentane Größe des Stapels und der Warteschlange:
   (a) **Stapel**: Ein leerer Stack S hat insgesamt 25 `push` Operationen durchgeführt, 12 `top` Operationen, und 10 `pop` Operationen, wovon 3 `null` zurückgegeben haben da der Stapel leer war. Wie groß ist der Stack S aktuell?
   (b) **Warteschlange**: Eine leere Warteschlange Q hat insgesamt 32 `enqueue` Operationen durchgeführt, 10 `first` Operationen, und 15 `dequeue` Operationen, wovon 5 `null` zurückgegeben haben da die Warteschlange leer war. Wie groß ist die Warteschlange Q aktuell?

[1] `https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html`, accessed 22.03.2022
[2] `https://docs.oracle.com/javase/7/docs/api/java/util/ArrayDeque.html`, accessed 22.03.2022

3. Wie sieht die leere Warteschlange W nach den folgenden Operationen aus und welche Werte werden ausgegeben?

   (a) enqueue(5)
   (b) enqueue(1)
   (c) print(first())
   (d) print(dequeue())
   (e) enqueue(2)
   (f) print(dequeue())
   (g) print(dequeue())
   (h) print(isEmpty())

Lösung:
Aufgabe 1: Deque

- Ausführen des ersten Prozesses in der Deque mittels *removeFirst*
- Thread teilt sich und wird vorne in der Deque mittels *addFirst* hinzugefügt
- *Stehlen* eines Threads von einem anderen Prozessor mittels *removeLast*

Aufgabe 2:

1. Stapel: 25 - (10-3) = 18 (Top only returns last item without removing it)
2. Warteschlange: 32 - 10 = 22 (first returns item without removing it)

Aufgabe 3:

1. nach a & b: [1,5]
2. c: [1,5]; ausgegeben: 5
3. d: [1]; ausgegeben: 5
4. e: [2,1]
5. f: [2], ausgegeben: 1
6. g: [], ausgegeben: 2
7. h: [], ausgegeben: true

Aufgabe 4 (4 Punkte): Türme von Hanoi

Die Türme von Hanoi sind ein mathematisches Knobel- und Geduldsspiel, in dem es darum geht, alle Teile des Turms von einem Ausgangspunkt zu einem Endpunkt zu bewegen [3]. Das Spiel besteht aus drei gleich großen Stäben A, B und C, auf die mehrere gelochte Scheiben gelegt werden, alle verschieden groß. Zu Beginn liegen alle Scheiben auf Stab A, der Größe nach geordnet, mit der größten Scheibe unten und der Kleinsten oben. Ziel des Spiels ist es, den kompletten Scheiben-Stapel von A nach C zu versetzen.

Die Regeln lauten wie folgt:

1. Es darf pro Zug nur eine Scheibe bewegt werden.

[3] `https://de.wikipedia.org/wiki/%C3%BCrme_von_Hanoi`, zugegriffen am 18.03.2022

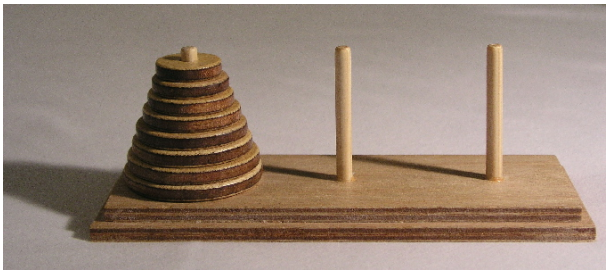


Abbildung 1: Tower of Hanoi(Source: `https://en.wikipedia.org/wiki/Tower_of_Hanoi#/media/File:Tower_of_Hanoi.jpeg`)

2. Bei jedem Zug darf die oberste Scheibe eines beliebigen Stabes auf die oberste Scheibe eines anderen Stapels gelegt werden.

3. Keine Scheibe darf auf eine kleinere Scheibe gelegt werden. Folglich sind zu jedem Zeitpunkt des Spieles die Scheiben auf jedem Feld der Größe nach geordnet.

Die folgende rekursive Funktion löst dieses Problem:

```
1 Algorithm TowerOfHanoi(n,s1,s2,s3):
2 if n = 1 then
3         s2.push(s1.pop()) //place disk from stack 1 to stack 2
4 else
5         TowerOfHanoi(n-1, s1, s3, s2) // put n-1 disks from stack 1 to stack 3
6         s2.push(s1.pop())  //put n-th disk from stack 1 to stack 2
7         TowerOfHanoi(n-1, s3, s2, s1)   //put n-1 disk from stack 3 to stack 2
```

Hinweis: `pop()` gibt das letzte/oberste Element des jeweiligen Stapels zurück. `push(e)` legt ein Element e auf den jeweiligen Stapel.

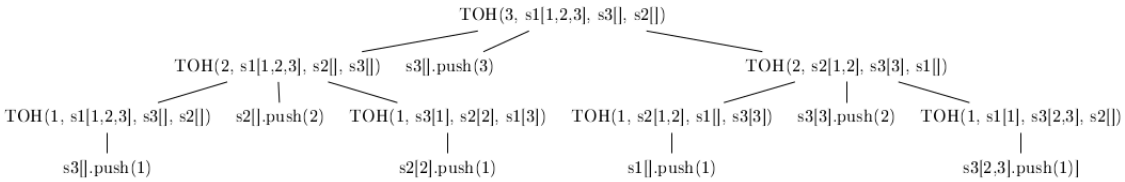
Bitte beantworte folgende Fragen:

1. Bestimme und zeichnen Sie den Aufrufbaum für den folgenden Funktionsaufruf auf. Um den Algorithmus auszuführen, werden n-Elemente auf den Stack `stack1` gepushed. Um diese Aufgabe zu lösen nehmen Sie an, dass die Elemente 3, 2, und 1 auf `stack1` gepushed wurden.

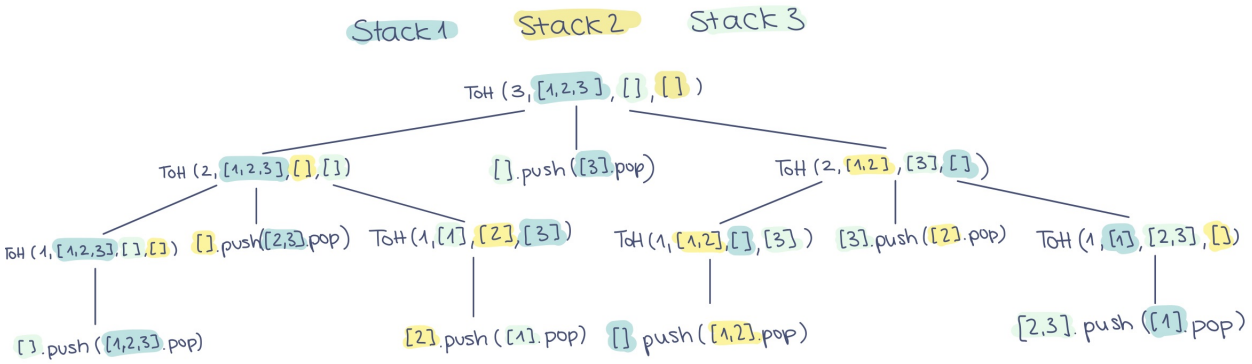
```
1         # Move the tower from stack one to stack three with stack two as
      temporary stack
2         TowerOfHanoi(len(stack1), stack1, stack3, stack2)
3
```

2. Welche Laufzeit-Komplexität angegeben in der Groß-O Notation hat diese angegebene rekursive Funktion - Erklären Sie Ihre Behauptung intuitiv und wählen dafür unterschiedlich kleine n-Elemente?

Lösung:



Und hier noch der Aufrufbaum aus der Besprechung im Proseminar:



Komplexität: exponentieller Algorithmus, $O(2^n)$
Bestimmung durch intuitive Weise: $2^n - 1$
n = 1 –> 1 Zug; n = 2 –> 3 Züge; n = 6 –> 63 Züge; ...