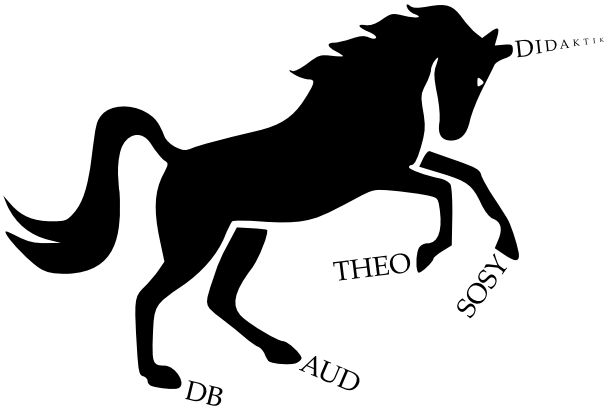
66115
Herbst
2014

Theoretische Informatik / Algorithmen (vertieft)

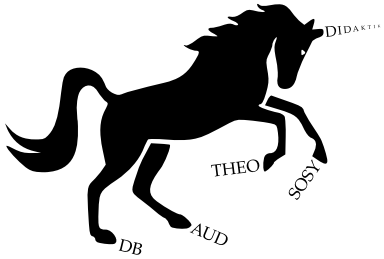
Aufgabenstellungen mit Lösungsvorschlägen



Die Bschlangaul-Sammlung

Hermine Bschlangaul and Friends

Aufgabenübersicht



Die Bschlangaul-Sammlung
Hermine Bschlangaul and Friends

Eine freie Aufgabensammlung mit Lösungen von Studierenden für Studierende zur Vorbereitung auf die 1. Staatsexamensprüfungen des Lehramts Informatik in Bayern.

Thema Nr. 2

Aufgabe 5 [Klasse „Stapel“ mit Methode „merge()“]

Gegeben sei eine Standarddatenstruktur Stapel (Stack) mit den Operationen

- `void push(Element e)`
- `Element pop()`,
- `boolean isEmpty()`.

sowie dem Standardkonstruktor `Stapel()`, der einen leeren Stapel zur Verfügung stellt.

(a) Geben Sie eine Methode `Stapel merge(Stapel s, Stapel t)` an, die einen aufsteigend geordneten Stapel zurückgibt, unter der Bedingung, dass die beiden übergebenen Stapel aufsteigend sortiert sind, ð `S.pop()` liefert das größte Element in `s` zurück und `T.pop()` liefert das größte Element in `t` zurück. Als Hilfsdatenstruktur dürfen Sie nur Stapel verwenden, keine Felder oder Listen.

Hinweis: Nehmen Sie an, dass Objekte der Klasse `Element`, die auf dem Stapel liegen mit `compareTo()` vergleichen werden können. Zum Testen haben wir Ihnen eine Klasse `StapelTest` zur Verfügung gestellt, sie können Ihre Methode hier einfügen und testen, ob die Stapel korrekt sortiert werden. Überlegen Sie auch, was geschieht, wenn einer der Stapel (oder beide) leer ist!

Lösungsvorschlag

```
  public static Stapel merge(Stapel s, Stapel t) {
    // Die beiden Stapel unsortiert aneinander hängen.
    Stapel mergedStack = new Stapel();
    while (!s.isEmpty()) {
      mergedStack.push(s.pop());
    }
    while (!t.isEmpty()) {
      mergedStack.push(t.pop());
    }
    // https://www.geeksforgeeks.org/sort-stack-using-temporary-stack/
    Stapel tmpStack = new Stapel();
    while (!mergedStack.isEmpty()) {
      Element tmpElement = mergedStack.pop();
      while (!tmpStack.isEmpty() && tmpStack.top.getValue() >
↪   tmpElement.getValue()) {
        mergedStack.push(tmpStack.pop());
      }
      tmpStack.push(tmpElement);
    }
    return tmpStack;
  }
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/Stapel.java`

Komplette Klasse Stapel

```java
/**
 * https://www.studon.fau.de/file2860857_download.html
 */
public class Stapel {
  public Element top;

  public Stapel() {
    top = null;
  }

  /**
   * @param element Das Element, dass hinzugefügt werden soll zur Stapel.
   */
  public void push(Element element) {
    element.setNext(top);
    top = element;
  }

  /**
   * @return Das Element oder null, wenn der Stapel leer ist.
   */
  public Element pop() {
    if (top == null) {
      return null;
    }
    Element element = top;
    top = top.getNext();
    return element;
  }

  /**
   * @return Wahr wenn der Stapel leer ist.
   */
  public boolean isEmpty() {
    return top == null;
  }

  /**
   * @param s Stapel s
   * @param t Stapel t
   *
   * @return Ein neuer Stapel.
   */
  public static Stapel merge(Stapel s, Stapel t) {
    // Die beiden Stapel unsortiert aneinander hängen.
    Stapel mergedStack = new Stapel();
    while (!s.isEmpty()) {
      mergedStack.push(s.pop());
    }
    while (!t.isEmpty()) {
      mergedStack.push(t.pop());
    }
    // https://www.geeksforgeeks.org/sort-stack-using-temporary-stack/
    Stapel tmpStack = new Stapel();
```

```
    while (!mergedStack.isEmpty()) {
      Element tmpElement = mergedStack.pop();
      while (!tmpStack.isEmpty() && tmpStack.top.getValue() >
↪   tmpElement.getValue()) {
        mergedStack.push(tmpStack.pop());
      }
      tmpStack.push(tmpElement);
    }
    return tmpStack;
  }

  public static void main(String[] args) {
    Stapel sa = new Stapel();
    sa.push(new Element(1));
    sa.push(new Element(2));
    sa.push(new Element(4));
    sa.push(new Element(5));
    sa.push(new Element(7));
    sa.push(new Element(8));
    Stapel sb = new Stapel();
    sb.push(new Element(2));
    sb.push(new Element(3));
    sb.push(new Element(6));
    sb.push(new Element(9));
    sb.push(new Element(10));

    Stapel sc = Stapel.merge(sa, sb);

    while (!sc.isEmpty()) {
      System.out.print(sc.pop().getValue() + ", ");
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/Stapel.java`

Komplette Klasse Element

```
/**
 * https://www.studon.fau.de/file2860856_download.html
 */
public class Element {
  public int value;

  public Element next;

  public Element() {
    this.next = null;
  }

  public Element(int value, Element element) {
    this.value = value;
```

```
    this.next = element;
  }

  public Element(int value) {
    this.value = value;
    this.next = null;
  }

  public int getValue() {
    return value;
  }

  public Element getNext() {
    return next;
  }

  public void setNext(Element element) {
    next = element;
  }

  public int compareTo(Element element) {
    if (getValue() > element.getValue()) {
      return 1;
    } else if (element.getValue() == getValue()) {
      return 0;
    } else {
      return -1;
    }
  }

}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/Element.java`

Test-Klasse

```
import static org.junit.Assert.*;
import org.junit.Test;

/**
 * https://www.studon.fau.de/file2860850_download.html
 */
public class StapelTest {

  @Test
  public void testeMethodenPushPop() {
    Stapel stapel = new Stapel();
    stapel.push(new Element(1));
    stapel.push(new Element(2));
    stapel.push(new Element(3));

    assertEquals(3, stapel.pop().value);
```

```
    assertEquals(2, stapel.pop().value);
    assertEquals(1, stapel.pop().value);
  }

  @Test
  public void testeMethodeMerge() {
    Stapel sa = new Stapel();
    sa.push(new Element(1));
    sa.push(new Element(3));
    sa.push(new Element(5));

    Stapel sb = new Stapel();
    sb.push(new Element(2));
    sb.push(new Element(4));

    Stapel sc = Stapel.merge(sa, sb);

    assertEquals(5, sc.pop().getValue());
    assertEquals(4, sc.pop().getValue());
    assertEquals(3, sc.pop().getValue());
    assertEquals(2, sc.pop().getValue());
    assertEquals(1, sc.pop().getValue());
  }

  @Test
  public void testeMethodeMergeMehrWerte() {
    Stapel sa = new Stapel();
    sa.push(new Element(1));
    sa.push(new Element(2));
    sa.push(new Element(4));
    sa.push(new Element(5));
    sa.push(new Element(7));
    sa.push(new Element(8));
    Stapel sb = new Stapel();
    sb.push(new Element(2));
    sb.push(new Element(3));
    sb.push(new Element(6));
    sb.push(new Element(9));
    sb.push(new Element(10));

    Stapel sc = Stapel.merge(sa, sb);

    assertEquals(10, sc.pop().getValue());
    assertEquals(9, sc.pop().getValue());
    assertEquals(8, sc.pop().getValue());
    assertEquals(7, sc.pop().getValue());
    assertEquals(6, sc.pop().getValue());
    assertEquals(5, sc.pop().getValue());
    assertEquals(4, sc.pop().getValue());
    assertEquals(3, sc.pop().getValue());
    assertEquals(2, sc.pop().getValue());
    assertEquals(2, sc.pop().getValue());
    assertEquals(1, sc.pop().getValue());
  }
```

```java
  @Test
  public void testeMethodeMergeBLeer() {
    Stapel sa = new Stapel();
    sa.push(new Element(1));
    sa.push(new Element(3));
    sa.push(new Element(5));

    Stapel sb = new Stapel();

    Stapel sc = Stapel.merge(sa, sb);

    assertEquals(5, sc.pop().getValue());
    assertEquals(3, sc.pop().getValue());
    assertEquals(1, sc.pop().getValue());
  }

  @Test
  public void testeMethodeMergeALeer() {
    Stapel sa = new Stapel();

    Stapel sb = new Stapel();
    sb.push(new Element(2));
    sb.push(new Element(4));

    Stapel sc = Stapel.merge(sa, sb);

    assertEquals(4, sc.pop().getValue());
    assertEquals(2, sc.pop().getValue());
  }

}
```

Code-Beispiel auf Github ansehen: `src/test/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/StapelTest.java`

(b) Analysieren Sie die Laufzeit Ihrer Methode.

Lösungsvorschlag

Best case: $\mathcal{O}(1)$ Worst case: $\mathcal{O}(n^2)$

Aufgabe 6 [Selectionsort]

Gegeben sei ein einfacher Sortieralgorithmus, der ein gegebenes Feld A dadurch sortiert, dass er das *Minimum* m von A *findet*, dann das Minimum von A ohne das Element m usw.

(a) Geben Sie den Algorithmus in Java an. Implementieren Sie den Algorithmus *in situ*, ðso, dass er außer dem Eingabefeld nur konstanten Extraspeicher benötigt. Es steht eine Testklasse zur Verfügung.

Lösungsvorschlag

```java
public class SortierungDurchAuswaehlen {
  static void vertausche(int[] zahlen, int index1, int index2) {
    int tmp = zahlen[index1];
```

```java
    zahlen[index1] = zahlen[index2];
    zahlen[index2] = tmp;
  }

  static void sortiereDurchAuswählen(int[] zahlen) {
    // Am Anfang ist die Markierung das erste Element im Zahlen-Array.
    int markierung = 0;
    while (markierung < zahlen.length) {
      // Bestimme das kleinste Element.
      // 'min' ist der Index des kleinsten Elements.
      // Am Anfang auf das letzte Element setzen.
      int min = zahlen.length - 1;
      // Wir müssen nicht bis letzten Index gehen, da wir 'min' auf das letzte
↪   Element
      // setzen.
      for (int i = markierung; i < zahlen.length - 1; i++) {
        if (zahlen[i] < zahlen[min]) {
          min = i;
        }
      }

      // Tausche zahlen[markierung] mit gefundenem Element.
      vertausche(zahlen, markierung, min);
      // Die Markierung um eins nach hinten verlegen.
      markierung++;
    }
  }

  public static void main(String[] args) {
    int[] zahlen = { 5, 2, 7, 1, 6, 3, 4 };
    sortiereDurchAuswählen(zahlen);
    for (int i = 0; i < zahlen.length; i++) {
      System.out.print(zahlen[i] + " ");
    }
  }
}
```

Code-Beispiel auf Github ansehen: `src/main/java/org/bschlangaul/examen/examen_66115/jahr_2014/herbst/SortierungDurchAuswaehlen.java`

(b) Analysieren Sie die Laufzeit Ihres Algorithmus.

Lösungsvorschlag

Beim ersten Durchlauf des *Selectionsort*-Algorithmus muss $n-1$ mal das Minimum durch Vergleich ermittel werden, beim zweiten Mal $n-2$. Mit Hilfe der *Gaußschen Summenformel* kann die Komplexität gerechnet werden:

$$(n-1)+(n-2)+\cdots+3+2+1=\frac{(n-1)\cdot n}{2}=\frac{n^2}{2}-\frac{n}{2}\approx\frac{n^2}{2}\approx n^2$$

Da es bei der Berechnung des Komplexität um die Berechnung der asymptotischen oberen Grenze geht, können Konstanten und die Addition, Subtraktion, Multiplikation und Division mit Konstanten z. b. $\frac{n^2}{2}$ vernachlässigt werden.

Der *Selectionsort*-Algorithmus hat deshalb die Komplexität $\mathcal{O}(n^2)$, er ist von der Ordnung $\mathcal{O}(n^2)$.

(c) Geben Sie eine Datenstruktur an, mit der Sie Ihren Algorithmus beschleunigen können.

Lösungsvorschlag

Der *Selectionsort*-Algorithmus kann mit einer Min- (in diesem Fall) bzw. einer Max-Heap beschleunigt werden. Mit Hilfe dieser Datenstruktur kann sehr schnell das Minimum gefunden werden. So kann auf die vielen Vergleiche verzichtet werden. Die Komplexität ist dann $\mathcal{O}(n \log n)$.