

# Datenstrukturen und effiziente Algorithmen

## Blatt 8

Markus Vieth, David Klopp, Christian Stricker

18. Dezember 2015



## Aufgabe 1

a)

Vor dem Einfügen der ersten Elements, werden head und tail aus 0 gesetzt. Anschließend wird das neue Element an der Stelle head eingefügt. Bei den folgenden Elementen wird Tail um 1 erhöht und wenn noch Platz in der Queue ist, wird das neue Element an der neuen Stelle auf die tail zeigt eingefügt. Bei Pop wird erst das Element auf das head zeigt gespeichert, dann wird der Platz auf null gesetzt. Anschließend wird head um 1 erhöht, damit es auf das nächste Element zeigt. Zum Schluss wird das gespeicherte Element zurückgegeben. Wird nun ein neues Element der Queue hinzugefügt werden und tail bereits auf das letzte Element im Array zeigen, so wird tail bei der +1 Operation durch eine Modulo-Operation mit der Größere des Arrays wieder auf den im letzten Schritt freien Anfang gesetzt. Dies kann beliebig fortgesetzt werden.

# Anhang

## 1 MapTest.java

```
1  class MyQueue<E> {

3     private final E[] queue;
4     private int head;
5     private int tail;
6     private int size;

8     //Erzeuge Objekt
9     @SuppressWarnings("unchecked")
10    public MyQueue ( int size) {
11        queue =(E[])new Object[size];
12        this.head = -1;
13        this.tail = -1;
14        this.size = 0;
15    }

18    public boolean push(E e) {

20        if ((tail+1)%queue.length == head) //Falls Tail direkt hinter Head, ist die Queue voll
21            return false;
22        else if (this.size() == 0) { //wenn leer, wurden die Pointer resetet
23            head++;
24            tail++;
25        }
26        else { //fange von vorne an, wenn hinten angekommen
27            tail = (tail+1)%queue.length;
28        }
29        queue[tail] = e;
30        size++;
31        return true;
32    }

35    public E pop() {
36        E value = null;
37        //Wenn Head gleich Tail, dann befindet sich nur ein Element in der Queue,
38        //nach dem pop kann diese resetet werden
39        if (head == tail) {
40            value = queue[head];
41            queue[head] = null; //Lösche verweise, damit Garbage Collector arbeiten kann
42            tail = -1;
43            head = -1;
44        } else if (size != 0) { //wenn ein Element vorhanden ist, nimm es und lösche es
45            value = queue[head];
46            queue[head] = null;
47            head = (head+1)%queue.length;
48        }
49        size--;
50        return value;
51    }

52    }

54    public int size() {
55        return size;
56    }

58    //Gibt Queue von Head bis Tail aus
59    public void print() {
60        for (int i = 0; i <this.size(); i++)
61            System.out.print(queue[(i+head)%queue.length]+" ");
62        System.out.println();
63    }
```

```

65 //Aufgabe 1 c)
66 public static void main (String... args) {
67     MyQueue<Integer> test = new MyQueue<Integer>(4);

69     test.push(4);
70     test.push(5);
71     test.push(0);
72     test.push(3);

74     test.pop();
75     test.pop();

77     test.push(2);
78     test.push(10);

80     test.print();

82 }
83 }

```

## 2 MyHashMap.java

```

1 import java.io.FileNotFoundException;
2 import java.io.FileReader;
3 import java.util.ArrayList;
4 import java.util.HashMap;
5 import java.util.HashSet;
6 import java.util.NoSuchElementException;
7 import java.util.Scanner;

9 class Puzzle {

11     int[][] arr; //damit die convert Methode einen Sinn hat
12     HashMap<Integer, ArrayList<Integer>> edges = new HashMap<Integer, ArrayList<Integer>>();
13     //HashMap mit möglichen Zügen
14     int n; //Kantenlänge
15     String start; //Ausgangssituation
16     private final int queueSize; //Gewählte Queue-Size

17     ///////////////////////////////////////////////////
18     //Aufgabe 2 b) Fortsetzung
19     ///////////////////////////////////////////////////
20     //Erstellt ein Puzzle-Objekt für eine Datei in filepath mit der Kantenlänge n
21     public Puzzle (String filepath, int n) throws FileNotFoundException {
22         this.n = n;
23         this.edges = createHash(n);
24         this.queueSize = fak(n*n);
25         Scanner input = new Scanner(new FileReader(filepath));
26         arr = new int[n][n];

28         //Zeilenweises einlesen der Datei
29         for (int i = 0; i < n; i++)
30             for (int j = 0; j < n; j++) {
31                 if (!input.hasNextInt()) {
32                     input.close();
33                     throw new NoSuchElementException("missing Element in Line " + (i*n));
34                 }
35                 arr[i][j] = input.nextInt();
36             }
37         input.close();
38         start = this.convert(arr);
39     }

41     ///////////////////////////////////////////////////
42     //Aufgabe 2 b)
43     ///////////////////////////////////////////////////
44     //Alternativer Aufruf für ein 8 Puzzle
45     public Puzzle (String filepath) throws FileNotFoundException{
46         this(filepath, 3);

```

```

47     }

49     //Alternativer Aufruf für ein bereits vorhandenes Puzzle
50     private Puzzle (String puzzle, int n, HashMap<Integer, ArrayList<Integer>> edges) {
51         start = puzzle;
52         this.n = n;
53         this.edges = edges;
54         this.queueSize = fak(n*n);
55     }

57     ////////////
58     //Aufgabe 2 a)
59     ////////////
60     //Konvertiert ein Puzzle in einen String
61     public String convert (int[][] arr) {
62         StringBuilder string = new StringBuilder();
63         for (int i = 0; i < arr.length; i++)
64             for (int j = 0; j < arr[i].length; j++)
65                 string.append(arr[i][j]);
66         return string.toString();
67     }

69     ////////////
70     //Aufgabe 2 d)
71     ////////////
72     //Prüft, ob ein gegebenes Puzzle lösbar ist, Algorithmus entspricht der Breitensuche aus der Vorlesung
73     public boolean loesbar () {
74         HashSet<String> visited = new HashSet<>(); //hier vorhandene Knoten sind grau
75         MyQueue<String> queue = new MyQueue<>(this.queueSize); //Die Queue
76         //HashMap<String, String> pi = new HashMap<>();

79         visited.add(start); //Makiere start grau
80         //pi.put(start, null);

82         queue.push(start); //platziere start in der queue
83         while(queue.size() != 0) {
84             String u = queue.pop();
85             int index = u.indexOf("0"); //Ermittle den freien Platz
86             ArrayList<Integer> edge = edges.get(index); //Ermittle die möglichen Züge/Kanten
87             for (int i = 0; i < edge.size(); i++) { //gehe alle Kanten ab
88                 String next = swap(u, index, edge.get(i)); //Erzeuge neue Nachbarknoten
89                 if (!visited.contains(next)) { //wenn Knoten weiß
90                     visited.add(next); //Setze Knoten grau
91                     //pi.put(next, u); //setze vorherigen Knoten als Vorgänger
92                     queue.push(next); //platziere Knoten in der Queue
93                 }
94                 if (this.correct(next)) //wenn Lösung erreicht
95                     return true;
96             }
97         } //Alle Knoten besucht
98         return false;
99     }

101     //Theoretisch eine Fakultätsfunktion
102     private static int fak(int n) {
103         /*if (n < 2)
104             return n;
105         return fak(n-1)*n;*/
106         return 362880; //9! um Rechenzeit zu sparen
107     }

109     ////////////
110     //Aufgabe 2 c)
111     ////////////
112     //hilfsmethode zur Erzeugung der Mashmap der möglichen Züge in einem Puzzle
113     private static HashMap<Integer, ArrayList<Integer>> createHash(int n) {

115         //Erstelle Hashmap
116         HashMap<Integer, ArrayList<Integer>> edges = new HashMap<Integer, ArrayList<Integer>>();

```

```

117     for (int i = 0; i < n; i++)
118         for (int j = 0; j < n; j++) {
119             ArrayList<Integer> temp = new ArrayList<Integer>();
120             edges.put(j+n*i, temp);
121             if (j == 0) {
122                 if (i == 0) { //Oben linke Ecke
123                     temp.add(j+1+n*i);
124                     temp.add(j+(i+1)*n);
125
126                 } else if( i == n-1) { //unten linke Ecke
127                     temp.add(j+(i-1)*n);
128                     temp.add(j+1+i*n);
129                 } else { //linker Rand
130                     temp.add(j+(i-1)*n);
131                     temp.add(j+1+i*n);
132                     temp.add(j+(i+1)*n);
133                 }
134             } else if (j == n-1) {
135                 if (i == 0) { //Oben rechte Ecke
136                     temp.add(j-1+n*i);
137                     temp.add(j+(i+1)*n);
138
139                 } else if( i == n-1) { //unten rechte Ecke
140                     temp.add(j+(i-1)*n);
141                     temp.add(j-1+i*n);
142                 } else {
143                     temp.add(j+(i-1)*n); //rechter Rand
144                     temp.add(j-1+i*n);
145                     temp.add(j+(i+1)*n);
146                 }
147             } else {
148                 if (i == 0) { //oberer Rand
149                     temp.add(j-1+n*i);
150                     temp.add(j+1+n*i);
151                     temp.add(j+(i+1)*n);
152
153                 } else if( i == n-1) { //unterer Rand
154                     temp.add(j+(i-1)*n);
155                     temp.add(j-1+i*n);
156                     temp.add(j+1+n*i);
157                 } else { //Mitte
158                     temp.add(j+(i-1)*n);
159                     temp.add(j-1+i*n);
160                     temp.add(j+1+n*i);
161                     temp.add(j+(i+1)*n);
162                 }
163             }
164         }
165     return edges;
166 }

167 //Ermittelt die Tiefe/Entfernung der Lösung
168 public int tiefe () {
169     //HashMap<String, Boolean> visited = new HashMap<>();
170     HashSet<String> visited = new HashSet<>();
171     MyQueue<String> queue = new MyQueue<>(this.queueSize);
172     //HashMap<String, String> pi = new HashMap<>();
173     HashMap<String, Integer> d = new HashMap<>(); //Distanzfunktion
174     //String loesung = null;
175     //visited.put(start, true);
176     visited.add(start);
177     //pi.put(start, null);
178     d.put(start, 0);
179
180     queue.push(start);
181     while(queue.size() > 0) {
182         String u = queue.pop();
183         int index = u.indexOf("0");
184         ArrayList<Integer> edge = edges.get(index);
185         for ( int i = 0; i < edge.size(); i++) {

```

```

187     String next = swap(u,index, edge.get(i));
188     if (!visited.contains(next)) {
189         visited.add(next);
190         //pi.put(next, u);
191         queue.push(next);
192         d.put(next, d.get(u)+1); //Länge des neuen Knoten ist Länge des Entdeckers +1
193     }
194     if (this.correct(next)) {
195         //loesung = next;
196         return d.get(next);
197         //result = Math.min(d.get(next), result);
198     }
199 }
200 }
201 //if (loesung == null)
202     return -1; //Gebe negativen Weg zurück, wenn kein Weg gefunden
203 //return d.get(loesung);
204 }

207 ///////////////
208 //Aufgabe 2 e)
209 ///////////////
210 //Ermittelt den Längsten Weg für ein n*n-1 Puzzle
211 public static int maxTiefe(int breite) throws FileNotFoundException {
212     int max = -1;
213     int n = breite*breite;

215     StringBuilder temp = new StringBuilder();
216     for (int i = 0; i < n; i++) //Erzeuge Ansatz
217         temp.append(i+"");
218     HashMap<Integer, ArrayList<Integer>> edges = createHash(breite); //Erzeuge einmalig die Hashmap
219     return maxTiefe("", temp.toString(), max, breite, n, edges);
220 }

222 //Vorisch extrem lange Laufzeit
223 private static int maxTiefe(String pre, String perm, int max, int breite, int n,
224     HashMap<Integer, ArrayList<Integer>> edges) throws FileNotFoundException {

226     if ( n == 0) { //Abbruchbedingung
227         int temp = new Puzzle(pre, breite, edges).tiefe();
228         return Math.max(temp, max);
229     }

231     int result = max; //setze bisheriges maximum als Startwert

233     for (int i = 0; i < n; i++) { //Bilde rekursiv alle Permutationen des Startstrings
234         result = Math.max(
235             maxTiefe(pre + perm.charAt(i), perm.substring(0, i) + perm.substring(i+1, n), result,
236                 breite, n-1, edges), result);
237     }
238     if (n > 4) //Fortschrittsanzeige
239         System.out.print("-");
240     return result;
241 }

243 // Hilfsfunktion zum tauschen zweier Zeichen in einem String
244 private static String swap (String string, int first, int last) {
245     char[] temp = string.toCharArray();
246     swap(temp, first, last);
247     return String.valueOf(temp);
248 }

250 // Hilfsfunktion zum tauschen zweier Zeichen in einem char[]
251 private static void swap (char[] arr, int first, int last) {
252     char temp = arr[first];
253     arr[first] = arr[last];
254     arr[last] = temp;
255 }

```



```
257 //Überprüft, ob der übergebene String eine gültige Lösung ist
258 private boolean correct (String solution) {
259     int l = solution.length();
260     if (!solution.endsWith("0"))
261         return false;
262     for ( int i = 1; i < l-1; i++) {
263         if ( Integer.parseInt(solution.charAt(i-1)+"") >= Integer.parseInt(solution.charAt(i)+""))
264             return false;
265     }
266     return true;
267 }

269 //Ein paar Test
270 public static void main (String... args) throws FileNotFoundException {
271     Puzzle test = new Puzzle("in.txt");
272     System.out.println(test.loesbar()); //Gibt für das Beispiel aus dem Blatt korrekt false zurück
273     System.out.println(test.tiefe());
274     System.out.println(Puzzle.maxTiefe(3)); //Sollte 31 zurückgeben
275 }
276 }
```