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 lösch es
45       value = queue[head];
46       queue[head] = null;
47       head = (head+1)%queue.length;
48     }
49     size--;
50     return value;

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>>();
    //Hashmap mit möglichen Zügen
13    int n;  //Kantenlänge
14    String start; //Ausgangssituation
15    private final int queueSize;  //Gewählte Queue-Size

17    //////////////
18    //Aufgabe 2 b) Fortsetzung
19    //////////////
20    //Erstellt ein Puzzle-Objekt für eine Datei in filpath 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 gegebens Puzzle lösbar ist, Algorithmus entsrpicht 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   //hilfmethode 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
168     //Ermittelt die Tiefe/Entfernung der Lösung
169     public int tiefe () {
170       if (!this.loesbar2())
171         return -1;
172
173       //HashMap<String, Boolean> visited = new HashMap<>();
174       HashSet<String> visited = new HashSet<>();
175       MyQueue<String> queue = new MyQueue<>(this.queueSize);
176       //HashMap<String, String> pi = new HashMap<>();
177       HashMap<String, Integer> d = new HashMap<>();  //Distanzfunktion
178       //String loesung = null;
179       //visited.put(start, true);
180       visited.add(start);
181       //pi.put(start, null);
182       d.put(start, 0);
183
184       queue.push(start);
185       while(queue.size() > 0) {
186         String u = queue.pop();
```

```
187       int index = u.indexOf("0");
188       ArrayList<Integer> edge = edges.get(index);
189       for ( int i = 0; i < edge.size(); i++) {
190         String next = swap(u,index, edge.get(i));
191         if (!visited.contains(next)) {
192           visited.add(next);
193           //pi.put(next, u);
194           queue.push(next);
195           d.put(next, d.get(u)+1);  //Länge des neuen Knoten ist Länge des Entdeckers +1
196         }
197         if (this.correct(next)) {
198           //loesung = next;
199           return d.get(next);
200           //result = Math.min(d.get(next), result);
201         }
202       }
203     }
204     //if (loesung == null)
205       return -1;  //Gebe negativen Weg zurück, wenn kein Weg gefunden
206     //return d.get(loesung);
207   }


210   //////////
211   //Aufgabe 2 e)
212   //////////
213   //Ermittelt den Längesten Weg für ein n*n-1 Puzzle
214   public static int maxTiefe(int breite) throws FileNotFoundException {
215     int max = -1;
216     int n = breite*breite;

218     StringBuilder temp = new StringBuilder();
219     for (int i = 0; i < n; i++)  //Erzeuge Ansatz
220       temp.append(i+"");
221     HashMap<Integer, ArrayList<Integer>> edges = createHash(breite);  //Erzeuge einmalig die Hashmap
222     return maxTiefe("", temp.toString(), max, breite, n, edges);
223   }

225   //Vorischt extrem lange Laufzeit (>9h)
226   private static int maxTiefe(String pre, String perm, int max,int breite, int n,
227       HashMap<Integer, ArrayList<Integer>> edges) throws FileNotFoundException {

229     if ( n == 0) {  //Abbruchbedinung
230       int temp = new Puzzle(pre, breite, edges).tiefe();
231       return Math.max(temp, max);
232     }

234     int result = max;  //setze bisheriges maxiumum als Startwert

236     for (int i = 0; i < n; i++) {  //Bilde rekursiv alle Permutationen des Startstrings
237       result = Math.max(
238           maxTiefe(pre + perm.charAt(i),perm.substring(0, i)+ perm.substring(i+1, n), result,
239               breite, n-1, edges), result);
240     }
241     if (n > 4 )  //Fortschrittsanzeige
242       System.out.print("-");
243     return result;
244   }

246   // Hilfsfunktion zum tauschen zweier Zeichen in einem String
247   private static String swap (String string, int first, int last) {
248     char[] temp = string.toCharArray();
249     swap(temp, first, last);
250     return String.valueOf(temp);
251   }

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

```
257       arr[last] = temp;
258     }
259
260     //Überprüft, ob der übergebene String eine gültige Lösung ist
261     private boolean correct (String solution) {
262       int l = solution.length();
263       if (!solution.endsWith("0"))
264         return false;
265       for ( int i = 1; i < l-1; i++) {
266         if ( Integer.parseInt(solution.charAt(i-1)+"") >= Integer.parseInt(solution.charAt(i)+""))
267           return false;
268       }
269       return true;
270     }
271
272     //Alternative, schnellere Variante für den lösbar Test
273     public boolean loesbar2() {
274       int n = 0;
275       for (int i = 0; i < start.length(); i++) {
276         int num = Integer.parseInt(start.charAt(i)+"");
277         for (int j = 0; j < i; j++) {
278           int num2 = Integer.parseInt(start.charAt(j)+"");
279           if (num2 > num)
280             n++;
281         }
282       }
283       int m = start.indexOf('0')+1;
284       int soll = start.length();
285       return (n+m)%2 == soll%2;
286     }
287
288     //Ein paar Test
289     public static void main (String... args) throws FileNotFoundException {
290       Puzzle test = new Puzzle("in.txt");
291       //System.out.println(test.loesbar2());
292       System.out.println(test.loesbar());  //Gibt für das Beispiel aus dem Blatt korrekt false zurück
293       System.out.println(test.tiefe());
294       System.out.println(Puzzle.maxTiefe(3));  //Sollte 31 zurückgeben
295     }
296   }
```