Datenstrukturen und effiziente Algorithmen
Blatt 7

Markus Vieth, David Klopp, Christian Stricker

11. Dezember 2015

Aufgabe 1

a) und b)

Als Kommentare im Quellcode im Anhang

c)

Quellcode im Anhang
Ja, die Behauptung, das sich die Laufzeit in Abhängigkeit zum Belegungsfaktor verändert konnte sich bestätigen:

Auszug aus der Laufzeitmessung (log.txt)

α	$T(20)$	$T_1(20)$	$T_2(20)$	$T_3(20)$
20	1.583.689 ns	670.281 ns	360.156 ns	553.252 ns
10	1.572.863 ns	112.055 ns	1.141.028 ns	319.780 ns
5	1.387.958 ns	112.932 ns	099.182 ns	1.175.844 ns
4	376.247 ns	119.661 ns	115.566 ns	141.020 ns
2	202.166 ns	039.497 ns	080.749 ns	081.920 ns
1	372.152 ns	098.597 ns	193.683 ns	079.872 ns
$0,5$	134.875 ns	026.624 ns	057.929 ns	050.322 ns
$0,1$	081.627 ns	020.187 ns	022.235 ns	039.205 ns
$0,05$	065.536 ns	011.118 ns	018.725 ns	035.693 ns

Anhang

1 MapTest.java

```
1  import java.util.Map;
2  import java.util.Scanner;
3  import java.io.FileInputStream;
4  import java.io.FileReader;
5  import java.io.IOException;
6  import java.io.InputStreamReader;
7  import java.io.PrintWriter;
8  import java.util.ArrayList;
9  import java.util.HashMap;
10 import java.util.TreeMap;

12 class Point implements Comparable<Point> {
13   public int x,y;

15   Point(int a,int b) {
16     x = a;
17     y = b;
18   }

20   @Override
21   public int hashCode() {
22     long z = x << 16 + y;
23     long p = 2147483647L; // Primzahl < 2^31
24     long a = 1189436865L; // zufällig 0 < a < p
25     long b = 1206511853L; // zufällig 0 <= b < p
26     // hier m = p

28     // Für ein m != p muss da dieser Stelle noch modulo m
29     // gerechnet werden
30     return (int)((a * z + b) % p);
31   }

33   @Override
34   public boolean equals(Object obj) {
35     Point p = (Point) obj;
36     if (x == p.x && y == p.y) return true;
37     return false;
38   }

40   @Override
41   public int compareTo(Point p) {
42     if (x < p.x) return -1;
43     if (x > p.x) return  1;
44     if (y < p.y) return -1;
45     if (y > p.y) return  1;
46     return 0;
47   }
48 };

50 public class MapTest {
51   public static void main(String[] args) throws IOException {


54 //    List<Point> P = new LinkedList<Point>();
55 //    P.add(new Point(1, 2));
56 //    P.add(new Point(0, 0));
57 //    P.add(new Point(1, 2));
58 //    P.add(new Point(0, 0));
59 //    P.add(new Point(1, 2));
60 //    P.add(new Point(0, 0));
61 //    P.add(new Point(1, 2));
62 //    P.add(new Point(0, 0));
63 //    P.add(new Point(1, 2));
64 //    P.add(new Point(0, 0));
```

```
65  //    P.add(new Point(2, 2));
66        ArrayList<Point[]> S = new ArrayList<Point[]>();

68        // Anlegen der HashMap, welche einen 2D Punkt auf eine Ganzzahl abbildet
69        HashMap<Point, Integer> H = new HashMap<Point, Integer>();

71        PrintWriter out = new PrintWriter("hash.txt");
72        Scanner in = new Scanner(new InputStreamReader(new FileInputStream("input.txt"),"UTF-8"));
73        // Fügt den Punkt p mit einem Index der HashMap zu, falls dieser noch
74        // nicht enthalten ist.
75        /*int i = 0;
76        for (Point p : P)
77          if (!H.containsKey(p))
78            H.put(p, i++);*/
79        int i = 0;
80        while (in.hasNextLine()) {
81          Point a = new Point(in.nextInt(), in.nextInt());
82          Point b = new Point(in.nextInt(), in.nextInt());
83          if (!H.containsKey(a))
84            H.put(a, i++);
85          if (!H.containsKey(b))
86            H.put(b, i++);
87          S.add(new Point[]{a,b});
88        }

90        // Gibt die (Key, Value) Paare aus, die oben hinzugefügt wurden
91        for (Map.Entry<Point, Integer> e : H.entrySet())
92          System.out.println(e.getKey().x + " " + e.getKey().y + " " + e.getValue());

94        //Aufgabe a)
95            //Hash
96        //1. Laufzeit: O(1)
97        System.out.println("Anzahl der Punkte: "+H.size());
98        //2. Laufzeit: O(n^2)
99        for (int j = 0; j < H.size(); j++) { // O(n)
100         for (Map.Entry<Point, Integer> e : H.entrySet()) // O(n)
101           if ( e.getValue().equals(j)) // O(1)
102             out.println(e.getKey().x+"  "+e.getKey().y); // O(1)
103       }
104       //3. O(n)
105       for ( int j = 0; j < S.size(); j++) {
106         out.println(H.get(S.get(j)[0])+", "+H.get(S.get(j)[1]));
107       }




112       out.close();
113       in.close();
114       // Selbes für die TreeMap
115       TreeMap<Point, Integer> T = new TreeMap<Point, Integer>();

117       out = new PrintWriter("tree.txt");
118       in = new Scanner(new FileReader("input.txt"));

120       // Fügt den Punkt p mit einem Index der TreeMap zu, falls dieser noch
121       // nicht enthalten ist.
122       i = 0;
123       /*for (Point p : P)
124         if (!T.containsKey(p))
125           T.put(p, i++);*/
126       while (in.hasNextLine()) {
127         Point a = new Point(in.nextInt(), in.nextInt());
128         Point b = new Point(in.nextInt(), in.nextInt());
129         if (!T.containsKey(a))
130           T.put(a, i++);
131         if (!T.containsKey(b))
132           T.put(b, i++);
133       }
```

```
135       // Gibt die (Key, Value) Paare aus, die oben hinzugefügt wurden
136       for (Map.Entry<Point, Integer> e : T.entrySet())
137         System.out.println(e.getKey().x + " " + e.getKey().y + " " + e.getValue());

139       //Aufgabe a)
140           //Tree
141       //1. O(1)
142       System.out.println("Anzahl der Punkte: "+T.size());
143       //2. O(n)
144       for (int j = 0; j < T.size(); j++) { // O(n)
145         for (Map.Entry<Point, Integer> e : T.entrySet()) // O(n)
146           if ( e.getValue().equals(j)) // O(1)
147             out.println(e.getKey().x+"  "+e.getKey().y); // O(1)
148       }
149       //3. O(log(n))
150       for ( int j = 0; j < S.size(); j++) {
151         out.println(T.get(S.get(j)[0])+", "+T.get(S.get(j)[1])); // O(log(n))
152       }



156       out.close();
157       in.close();


160     }
161   }
```

2 MyHashMap.java

```
1   import java.io.FileInputStream;
2   import java.io.FileNotFoundException;
3   import java.io.InputStreamReader;
4   import java.io.PrintWriter;
5   import java.io.UnsupportedEncodingException;
6   import java.text.DecimalFormat;
7   import java.util.ArrayList;
8   import java.util.HashSet;
9   import java.util.LinkedList;
10  import java.util.Random;
11  import java.util.Scanner;

13  class Point implements Comparable<Point> {
14    public int x,y;
15    static Random random = new Random();
16    static final long p = 2147483647L; // Primzahl < 2^31
17    static long a = nextLong(random,p-1)+1; // zufällig 0 < a < p
18    static long b = nextLong(random,p); // zufällig 0 <= b < p
19    static long m = 10000;

21    public static void newHash(int m) {
22      a = nextLong(random,p-1)+1; // zufällig 0 < a < p
23      b = nextLong(random,p); // zufällig 0 <= b < p
24      Point.m = m;
25    }

27    Point(int a,int b) {
28      x = a;
29      y = b;
30    }

32    private static long nextLong(Random rndm, long n) {
33      // error checking and 2^x checking removed because not necessary.
34      // modified version of nextInt(int range)
35      long bits, val;
36      do {
37        bits = (rndm.nextLong() << 1) >>> 1;
38        val = bits % n;
39      } while (bits-val+(n-1) < 0L);
```

```
40      return val;
41    }

43    @Override
44    public int hashCode() {
45      // randomisiere mich
46      long z = x << 16 + y;

48      if (((a * z + b) % p)%m < 0)
49        return (int)((((a * z + b) % p)%m)+m);
50      return (int)(((a * z + b) % p)%m);
51    }

53    @Override
54    public boolean equals(Object obj) {
55      Point p = (Point) obj;
56      if (x == p.x && y == p.y) return true;
57      return false;
58    }

60    @Override
61    public int compareTo(Point p) {
62      if (x < p.x) return -1;
63      if (x > p.x) return  1;
64      if (y < p.y) return -1;
65      if (y > p.y) return  1;
66      return 0;
67    }
68  };


71  // Hilfklasse
72  class Entry<K, V> {
73    private K key;
74    private V value;

76    public Entry(K key, V value) {
77      this.key = key;
78      this.value = value;
79    }

81    public K getKey()
82    {
83      return this.key;
84    }

86    public V getValue()
87    {
88      return this.value;
89    }

91    public void setValue(V value) {
92      this.value = value;
93    }
94  }


97  /*
98   * Da der Zugriff auf einen Index in einem Array in O(1) funktioniert, kann eine LinkedList in O(1)
99   * erhalten werden. Das Iteririeren über die gesamte Liste benötigt höchstens O(Anzahl Kollisionen
100  * für dieses Element)
101  *
102  *
103  */


106 public class MyHashMap<K, V> {
107   // nicht effizient, aber so kann die Abfrage in O(1) erfolgen
108   private HashSet<Entry<K, V>> entrySet = new HashSet<Entry<K, V>>();
109   // nicht schön aber Java erlaubt keine Arrays von generischen Typen
```

```
110   private Object[] values = new Object[65536];
111   private int size = 0;
112   public int n = 0; //Anuahl der Kollisionen

114   public void put(K key, V value) {
115     int index = key.hashCode();

117     // Wir können uns an dieser Stelle sicher sein, dass das Objekt eine
118     // LinkedList mit entsprechendem Entry ist,
119     // da wir keinen Zugriff von außen auf das Array erlauben
120     @SuppressWarnings("unchecked")
121     LinkedList<Entry<K, V>> list = (LinkedList<Entry<K, V>>) this.values[index];

123     // Wenn die Liste noch nicht existiert
124     if (list == null) {
125       list = new LinkedList<Entry<K, V>>();
126       this.values[index] = list;
127       n--;
128     }

130     // Wenn das Element mit dem gegeben Schlüssel existiert, dann überschreiben den Wert
131     boolean containsKey = false;
132     for (Entry<K, V> e : list) {
133       if (e.getKey().equals(key)) {
134         e.setValue(value);
135         containsKey = true;
136         break;
137       }
138     }

140     // füge an das Ende der Liste ein Entry Element bestehend aus Wert und Punkt ein, wenn
141     // das Element noch nicht existiert.
142     if (containsKey == false) {
143       Entry<K, V> item = new Entry<K, V>(key, value);
144       list.add(item);
145       entrySet.add(item);
146       n++;
147     }

149     size ++;
150   }


153   public HashSet<Entry<K, V>> entrySet() {
154     // O(1)
155     return this.entrySet;
156   }

158   public int size() {
159     // O(1)
160     return size;
161   }


164   public V get(K key) {
165     int index = key.hashCode();

167     @SuppressWarnings("unchecked")
168     LinkedList<Entry<K, V>> list = (LinkedList<Entry<K, V>>) this.values[index];

170     // Wenn die liste existiert, iteriere über alle Elemente der Liste und vergleiche die Objekte
171     if (list != null)
172       for (Entry<K, V> e : list)
173         if (e.getKey().equals(key))
174           return e.getValue();

176     // nichts gefunden
177     return null;
178   }
```

```
180   public boolean containsKey(K key) {
181     return this.get(key) != null;

183   }


186   public Point remove(Point p) {
187     int index = p.hashCode();

189     // Element suchen und entfernen
190     // ähnlich zu get
191     @SuppressWarnings("unchecked")
192     LinkedList<Entry<K, V>> list = (LinkedList<Entry<K, V>>) this.values[index];
193     if (list != null)
194       for (Entry<K, V> e : list)
195         if (e.getKey().equals(p)) {
196           size --;
197           list.remove(e);
198           return p;
199         }
200     return null;
201   }


204   public static void main(String[] args) throws FileNotFoundException,
205       UnsupportedEncodingException {

207     PrintWriter log = new PrintWriter("log.txt");

209     for (int m = 1; m <= 400; m++) {
210       Point.newHash(m);
211       MyHashMap<Point, Integer> h = new MyHashMap<Point, Integer>();

213       ArrayList<Point[]> s = new ArrayList<Point[]>();
214       Scanner in = new Scanner(new InputStreamReader(new FileInputStream("input.txt"),"UTF-8"));

216       int i = 0;
217       while (in.hasNextLine()) {
218         Point a = new Point(in.nextInt(), in.nextInt());
219         Point b = new Point(in.nextInt(), in.nextInt());
220         if (!h.containsKey(a))
221           h.put(a, i++);
222         if (!h.containsKey(b))
223           h.put(b, i++);
224         s.add(new Point[]{a,b});
225       }

227       // Analyse
228       PrintWriter out = new PrintWriter("myHash.txt");
229       //1. Laufzeit: O(1)
230       long start1 = System.nanoTime();
231       System.out.println("Anzahl der Punkte: "+h.size());
232       long end1 = System.nanoTime();
233       //2. Laufzeit: O(n^2)
234       long start2 = System.nanoTime();
235       for (int j = 0; j < h.size(); j++) { // O(n)
236         for (Entry<Point, Integer> e : h.entrySet()) // O(n)
237           if (e.getValue().equals(j)) // O(1)
238             out.println(e.getKey().x+"  "+e.getKey().y); // O(1)
239       }
240       long end2 = System.nanoTime();
241       //3. O(n)
242       long start3 = System.nanoTime();
243       for ( int j = 0; j < s.size(); j++) {
244         out.println(h.get(s.get(j)[0])+", "+h.get(s.get(j)[1]));
245       }
246       long end3 = System.nanoTime();

248       out.close();
249       in.close();
```

```
251         // Bonusaufgabe
252         DecimalFormat format1 = new DecimalFormat("#000,000;(#)");
253         DecimalFormat format2 = new DecimalFormat("#0.0000000;(#)");
254         long timeInNs1 = (end1-start1); // Laufzeit für Teilaufgabe a
255         long timeInNs2 = (end2-start2); // Laufzeit für Teilaufgabe b
256         long timeInNs3 = (end3-start3); // Laufzeit für Teilaufgabe c
257         double a = h.size*1.0/m;    // Belegungsfaktor
258         System.out.println("Belegungsfaktor: "+format2.format(a)+" | T(20): "
259             +format1.format(timeInNs1+timeInNs2+timeInNs3)+ " | T1(20): "
260             +format1.format(timeInNs1)+ " | T2(20): "+format1.format(timeInNs2)
261             + " | T3(20): "+format1.format(timeInNs3));
262         log.println("Belegungsfaktor: "+format2.format(a)+" | T(20): "
263             +format1.format(timeInNs1+timeInNs2+timeInNs3)+ " | T1(20): "
264             +format1.format(timeInNs1)+ " | T2(20): "+format1.format(timeInNs2)
265             + " | T3(20): "+format1.format(timeInNs3));
266       }
267       log.close();
268     }
269 }
```