Übung: Algorithmen und Datenstrukturen SS 2007

Prof. Lengauer
Sven Apel, Michael Claßen, Christoph Zengler, Christof König

Blatt 9

– Votierung in der Woche vom 02.07.07–06.07.07 –

Aufgabe 26 Vergleich von Suchverfahren

Im Skript finden sich auf Seite 68 und 69 zwei Übersichtstabellen zu den Laufzeiten einzelner Operationen. Die folgenden Aussagen sind diesen beiden Tabellen entnommen. Begründen Sie diese jeweils. Falls es sich um einen *worst case* handelt, reicht zur Begründung die Angabe eines Beispiels.

(a) Im *average case* benötigt eine erfolgreiche Suche in einer nicht sortierten Liste $(n+1)/2$ Schritte.

(b) Im *failure case* benötigt die Suche in einer nicht sortierten Liste im Durchschnitt $n+1$ Schritte.

(c) Im *failure case* benötigt die Suche in einer sortierten Liste im Durchschnitt $(n+2)/2$ Schritte.

(d) In einem binären Suchbaum ist der *worst case* für das Finden eines Elements n Schritte.

(e) Sei n die Anzahl der Werte in einer Hash-Tabelle mit m Slots. Warum braucht man zum Finden des Minimums/Maximums m Schritte? Handelt es sich dabei um eine *worst* oder *best case* Abschätzung?

(f) Was ist mit left-height bzw. right-height für die Laufzeit zum Finden des Minimums/Maximums in einem Suchbaum gemeint? Wie sieht damit die *worst*, *average* und *best case* Laufzeit genau aus?

Aufgabe 27 Hashing

Gegeben ist folgender Code einer Klasse Point:

```
public class Point {

   public int x;

   public int y;

   public Point(int x, int y) {
      this.x = x;
      this.y = y;
   }

   public String toString() {
      return "(" + x + "/" + y + ")";
   }

   public boolean equals(Object o) {
      if (o instanceof Point) {
         Point p = (Point) o;
         return p.x == this.x && p.y == this.y;
      }
      return false;
   }
}
```

Diese Punkte sollen in ein Array mittels eines Hashing-Verfahrens einsortiert werden. Dazu gibt es die Klasse `PointArray`. Der Konstruktor dieser Klasse legt ein neues PointArray einer gegebenen Größe an. Die Funktion `getHash(Point p)` liefert zu einem Punkt p den entsprechenden Hashwert. Die Methode `printArray()` gibt das PointArray aus.
Das Codefragment hierzu sieht wie folgt aus:

```
public class PointArray{

   private Point[] points;

   public PointArray(int length) {
      points = new Point[length];
   }

   private int getHash(Point p) {
      return (p.x + p.y) % points.length;
   }

   public void printArray() {
      for (int i=0; i<points.length-1; i++) {
         System.out.print(points[i] + " - ");
      }
```

```
      System.out.print(points[points.length-1] + "\n");
   }
}
```

a) Implementieren Sie eine Methode `insertPoint(Point p)`, die einen gegebenen Punkt p in das Array einfügt. Die Position des Punktes im Array soll dabei über die Hashfunktion ermittelt werden. Bei einer Kollision sollen Sie die Strategie des *linear chaining* benutzen. Behandeln Sie auch den Fall, dass das Array voll ist.

b) Implementieren Sie eine Methode `getPoint(Point p)`, die einen gegebenen Punkt p im Array sucht. Sie soll den Punkt zurückgeben, falls sie ihn gefunden hat. Findet sie den Punkt nicht, soll sie `null` zurückgeben. Die Funktion soll dabei natürlich nur so weit wie nötig suchen, und nicht jedesmal das ganze Array durchlaufen.

c) Testen Sie Ihre Methoden mit der folgenden Testklasse und verifizieren Sie die Ergebnisse:

```
public class Test {

   public static void main(String[] args) {
      PointArray points = new PointArray(10);

      Point p1 = new Point(4,4);
      Point p2 = new Point(5,3);
      Point p3 = new Point(7,8);
      Point p4 = new Point(9,1);
      Point p5 = new Point(10,11);
      Point p6 = new Point(14,11);
      Point p7 = new Point(1,1);

      points.insertPoint(p1);
      points.insertPoint(p2);
      points.insertPoint(p3);
      points.insertPoint(p4);
      points.insertPoint(p5);
      points.insertPoint(p6);
      points.insertPoint(p7);

      System.out.println("Array: ");
      points.printArray();

      System.out.println("\nSearches: ");
      System.out.println("4,4: " + points.getPoint(new Point(4,4)));
      System.out.println("14,11: " + points.getPoint(new Point(14,11)));
      System.out.println("3,0: " + points.getPoint(new Point(3,0)));
```

```
      System.out.println("1,1: " + points.getPoint(new Point(1,1)));
      System.out.println("8,0: " + points.getPoint(new Point(8,0)));
   }
}
```

Anmerkung: Den Code Der Klassen `Point` und `Test` finden Sie im Stud.IP und auf der Vorlesungs-Hompage. Ebenso das Codefragment der Klasse `PointArray`.

Aufgabe 28 Charakterisierungen von Bäumen

Sei $G = (V, E)$ ein einfacher ungerichteter Graph mit n Knoten und m Kanten. Zeigen Sie, dass folgende Aussagen äquivalent sind:

1. G ist zusammenhängend und $m = n - 1$.
2. G ist kreisfrei und $m = n - 1$.
3. Jede weitere Kante erzeugt einen (den ersten) Kreis.
4. Zwischen je zwei Knoten u, v gibt es genau einen einfachen Weg.