Datenstrukturen & Algorithmen

Dario Marti 18-109-967
Nûr Johanna Studer 12-122-438

Serie 5

Theoretische Aufgaben

1. Reihenfolge umkehren

Nicht rekursiv, Zeit $\Theta(n)$, einfach verkettet, n Elemente

```
LIST-REVERSE(L)
      point := NIL                    //head wird zu tail werden und auf NIL zeigen
      current := L.head               //aktuelle Stelle, Beginn bei Head
      WHILE current != nil
            next := current.point     //Zwischenspeicher Zeiger von akt. El.
            current.point := point    //Zeiger überschrieben (1. NIL)
            point := current          // Zeiger von nächstem El. (akt. El.)
            current := next           // auf nächstes Element
      L.head = point                  //Kopf = letztes El. Auf das gezeigt wurde
```

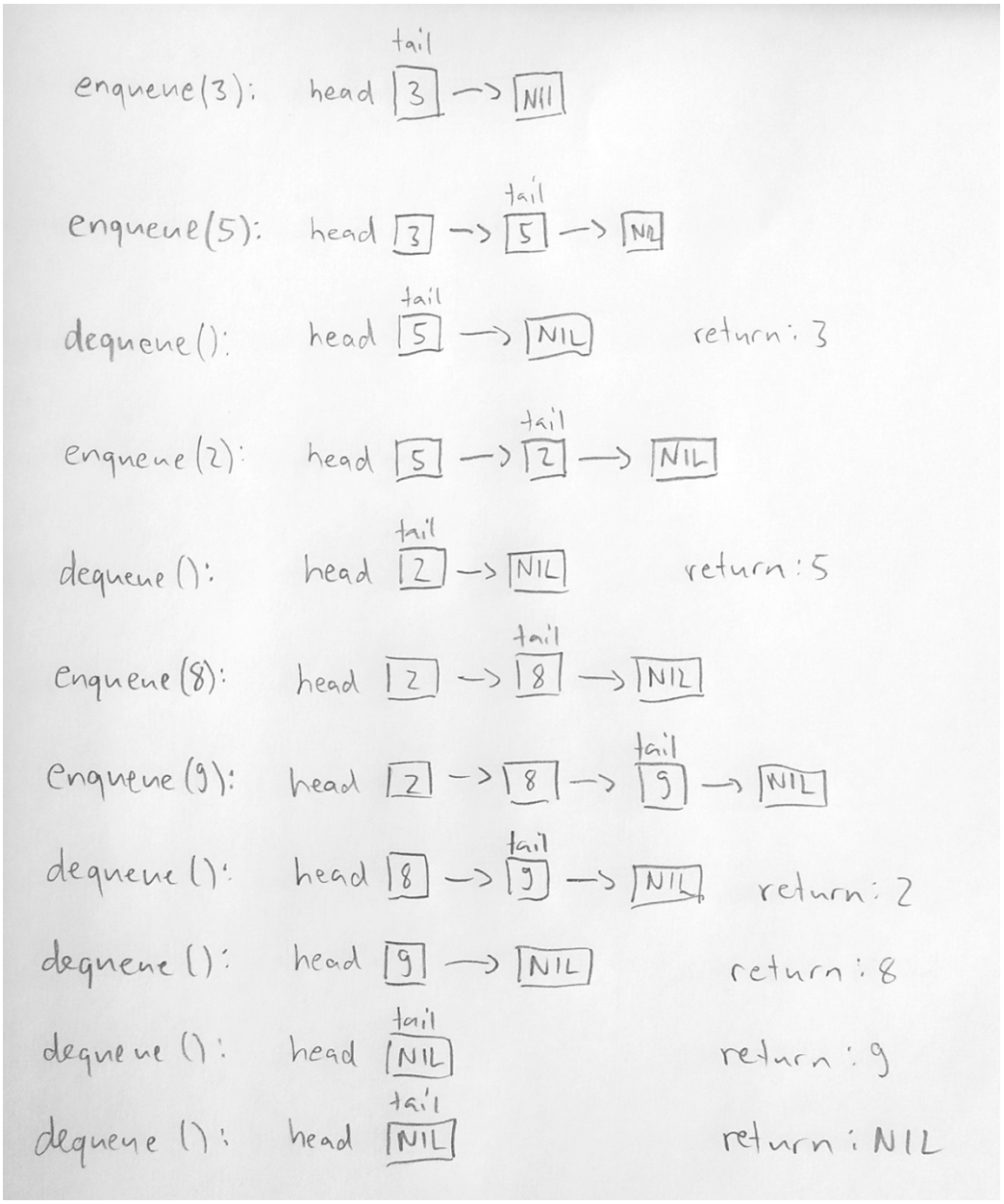
2. Warteschlange

e.next ist ein Element oder NIL, e.key ist ein Wert
Nil.next und NIL.key geht nicht

```
Enqueue(Queue Q, Element e)
      e.next := NIL                   //neues Element zeigt auf NIL
      If Q.head = NIL                 //e ist erstes Element
            Q.head := e
      Else
            Q.tail.next := e          //aktueller tail zeigt auf e
      Q.tail := e                     //e wird tail


Dequeue(Queue Q)
      If Q.head = NIL                 //Q.head.key geht nicht wenn Q.head = NIL
            Return NIL
      ELSE                            //Queue nicht leer
      e := Q.head.key                 //Zwischenspeicher Ausgabe (e kein Element)
      If Q.head.next = NIL            //letztes Element
            Q.tail := NIL             //tail = NIL
      Q.head = Q.head.next            //Element oder NIL
      return e
```

Dario Marti 18-109-967
Nûr Johanna Studer 12-122-438



3. Schlüssel aller Knoten eines gerichteten Baums: rekursiv

```
GET-KEYS(n)
IF n == NIL
      RETURN
ELSE
      PRINT(n.key)
      GET-KEYS (n.left-child)
      GET-KEYS (n.right-sibling)
```

4. Schlüssel aller Knoten eines gerichteten Baums: nicht rekursiv

```
S := NEW STACK();
S.PUSH(root)

TRAVERSE_TREE(Node: root)
WHILE S != NIL
current := Node
     WHILE current != NIL
          S.PUSH(Node.left_child)
          current := current.left_child
     END

     last_node := S.POP()
     PRINT(last_node.key)
END
```

5. Merge

```
MERGE (L, R, M)

M.head := NIL

IF L.head != NIL | R.head != NIL

     IF (L.head.key <= R.head.key) | (R.head = NIL && L.head != NIL)
          M.head := L.head
          L.head := L.head.next
     ELSE
          M.head := R.head
          R.head := R.head.next

     M.tail := M.head.next

WHILE L.head != NIL | R.head != NIL
     IF (L.head.key <= R.head.key) | (R.head = NIL && L.head != NIL)
          M.tail := L.head
          M.tail := M.tail.next
          L.head := L.head.next
     ELSE
          M.tail := R.head
          M.tail := M.tail.next
          R.head := R.head.next

M.tail.next := M.head      //zyklisch
Return M
```

Praktische Aufgaben

Die Lösungen zu den Aufgaben 1 und 2 befinden sich im File KDTreeVisualization.java.
Die 3 konnten wir nicht lösen.