Datenstrukturen und Effiziente Algorithmen
Fachbereich IV - Informatik
Universität Trier

Moritz Gobbert
gobbert@informatik.uni-trier.de
Raum H 413

Algorithmen und Datenstrukturen

Sommersemester 2016
Aufgabenblatt 4
Abgabe: Montag, 23. Mai 2016, 10:00 Uhr

Aufgabe 4.1 *(8 Punkte)*

Implementieren Sie eine doppelt verkettete Liste in Pseudo-Code mit den folgenden Operationen:

- `insert(Listelem` *pos*`, Integer` *x*`)` – Diese Operation legt ein `Listelem` mit Wert *x* als Nachfolger des Elements *pos* an.
- `delete(Listelem` *elem*`)` – Diese Operation löscht das Listenelement *elem*
- `find(integer` *x*`)` – diese Operation durchsucht die Liste und liefert ein Element mit Wert *x* zurück, wenn dieses vorhanden ist. Sonst gibt die Funktion „Null“ zurück.

(*Hinweis:* Sie dürfen ruhig annehmen, dass bei den ersten beiden Operationen die der Funktion übergebenen Listenelemente auch tatsächlich in der Liste vorhanden sind – Sie müssen also keine Ausnahmeregeln schreiben, für den Fall, dass *pos* bzw. *elem* nicht vorhanden sind.)

Lösung

```
1: class Listelem(int x):
2:     int value ← x
3:     Listelem prev ← Null
4:     Listelem next ← Null
5: end
```

```
1: class list():
2:     Listelem front ← Null
3:     Listelem back ← Null                    ▷ Ist nicht zwingend notwendig.
4: end
```

```
1: procedure insert(Listelem pos, int x)
2:     elem ← new Listelem(x)                              ▷ Element anlegen
3:     elem.next ← pos.next          ▷ Verweise des neuen Elements anpassen
4:     elem.prev ← pos
5:     pos.next ← elem                                  ▷ Alte Verweise anpassen
6:     if elem.next ≠ Null then      ▷ Wenn elem nicht das letzte Element ist
7:         (elem.next).prev ← elem
8:     else
9:         back ← elem                    ▷ sonst ist elem das letzte Element
```

```
    end if
end procedure
procedure DELETE(Listelem elem)
    if elem.prev ≠ NULL then          ▷ Wenn elem nicht das erste Element ist
        (elem.prev).next ← elem.next
    else                                ▷ sonst ist elem das erste Element
        front ← elem.next
    end if
    if elem.next ≠ NULL then          ▷ Wenn elem nicht das letzte Element ist
        (elem.next).prev ← elem.prev
    else                                ▷ sonst ist elem das letzte Element
        back ← elem.prev
    end if
    elem.next ← NULL
    elem.prev ← NULL
end procedure
function FIND(int x)
    Listelem elem ← front
    while elem ≠ NULL do
        if elem.value = x then
            return elem
        end if
        elem ← elem.next
    end while
    return NULL
end function
```

Aufgabe 4.2 *(10 Punkte)*

Schreiben Sie ein Programm (in Pseudo-Code) zur Implementierung der *dynamischen* Variante einer Queue/Schlange für ganze Zahlen. Dynamisch heißt in diesem Fall, dass die Queue beliebig viele Elemente speichern kann. Die Implementierung soll die folgenden Operationen unterstützen:

- `isEmpty()` – testet ob die Queue leer ist
- `append(int` x`)` – hängt ein neues Element mit Wert x an die Queue an
- `pop()` – entfernt das erste Element der Queue und gibt seinen Wert aus
- `top()` – gibt den Wert des ersten Elements aus

Verwenden Sie hierzu die einfach verkettete Liste aus der Vorlesung.

Lösung

```
class intQueue():
    Listelem front ← Null
    Listelem back ← Null
end
function isEmpty()
```

```
2:      if front = Null then
3:          return True
4:      end if
5:      return False
6:  end function
1:  procedure append(int x)
2:      elem ← new Listelem(x)
3:      if front = Null then
4:          front ← elem
5:          back ← elem
6:      else
7:          back.next ← elem
8:          back ← elem
9:          elem.next ← Null                ▷ Nicht unbedingt notwendig
10:     end if
11: end procedure
1:  function pop()
2:      if front ≠ Null then
3:          tmp ← front
4:          front ← front.next
5:          if front =Null then             ▷ Wenn die Liste nun leer ist
6:              back ← Null
7:          end if
8:          return tmp.value
9:      end if
10:     return Null
11: end function
1:  function top()
2:      if front ≠ Null then
3:          return front.value
4:      end if
5:      return Null
6:  end function
```