

Lösungen zum Thema Datenstrukturen

(1) (a) $n=0$: / $n=1$: 1 $n=2$: 1 $n=3$: 1

|

2 2 3 2 3

$n=4$:

1
/ \
2 3
/
4

1
/ \
2 4
/
3

1
/ \
3 2
/
4

1
/
2
/ \
3 4

1
/
2
/ \
4 3

(b) Alle Heaps für $n \leq 3$, bei $n = 4$ die ersten drei Heaps.

(c)

1	2	6	3	⊥	8	7	⊥	⊥	⊥	⊥	9
---	---	---	---	---	---	---	---	---	---	---	---

(2) (a) Das Element mit dem Schlüssel 18 verletzt die Heap-Eigenschaft, da es kleiner als 26 und 21 ist, aber weiter unten als diese im Heap steht.

(b)

12
/ \
21 19
/ \
26 99 30 24
/ \
32 18 101 128

->

12
/ \
21 19
/ \
18 99 30 24
/ \
32 26 101 128

->

12
/ \
18 19
/ \
21 99 30 24
/ \
32 26 101 128

(3) Bemerkung: Die Lösungen hängen natürlich von der gewählten Datenstruktur ab. Dies ist nur eine mögliche Lösung.

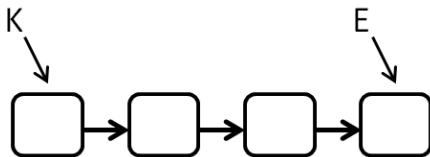
(a) Doppelt verkettete Liste mit zusätzlichem Attribut *min*.

(b) `add(L, x) //füge x in die Liste L ein`
 `// hier wäre der Code für das reine Einfügen in die Liste`
 `if x < L.min`
 `L.min = x`

(c) `minimum(L) //gib Minimum aus L zurück`

```
return L.min
```

- (d) Falls das zu löschende Element das minimale war, muss die komplette Liste nach dem neuen Minimum durchsucht werden. Die Laufzeit ist $\Theta(n)$. Bei einer normalen doppelt verketteten Liste wäre das Löschen (bei Kenntnis des Pointers) aber in konstanter Zeit möglich!
- (4) (a) Lösungsvorschlag: Hashtabelle mit Verkettung - Operationen laufen bei hinreichender Größe des Feldes in $O(1)$, Speicherplatz ist nicht beschränkt (außer durch Größe des Speichermediums).
- (b) Lösungsvorschlag: Warteschlange (Queue) - durch die FiFo-Strategie ändert sich die Reihenfolge der Anfragen beim Puffern nicht, außerdem werden ENQUEUE und DEQUEUE in jeweils konstantem Zeitaufwand ausgeführt.
- (c) Lösungsvorschlag: Max-Heap, da hier sowohl die Auswahl des Elements mit dem größten Schlüssel als auch das Einfügen neuer Prozesse hier besonders schnell ($O(\log(n))$) erfolgen kann.
- (5) Skizze:



Anm: K muss auf den Listenanfang zeigen, da nur hier das Löschen in konstanter Zeit möglich ist.

- (6) *ENQUEUE(x)*
- ```
01 if (E == NIL)
02 E = x;
03 else
04 E.next = x;
05 E = E.next;
```
- DEQUEUE()*
- ```
01  if (K == NIL)
02      error("Underflow!");
03  else
04      x = K;
05      K = K.next;
06  return x;
```