Datenstrukturen und Algorithmen

Vorlesung 3

Überblick

- Vorige Woche:
  - Arrays
  - Iteratoren

- Heute betrachten wir:
  - ADT Bag & SortedBag
  - ADT Set & SortedSet

ADT Bag / MultiSet

(Besprochen im Seminar 1)

- Domäne:

  $\mathcal{B}$ = {b | b ist ein Bag mit Elementen vom Typ TElem}

- ADT Bag ist ein Container, in welcher Elemente nicht eindeutig sind und keine Positionen haben (ein Element kann vielfach enthalten sein)

ADT Bag - Interface

- init(b)
  - pre: true
  - post: *b* ∈ $\mathcal{B}$, *b* ist ein leeres Bag

- add(b, e)
  - pre: *b* ∈ $\mathcal{B}$, *e* ∈ *TElem*
  - post: *b'*∈ $\mathcal{B}$, *b'*=*b* ∪ {e} (TElem *e* wird in dem Bag eingefügt)

- remove(b, e)
  - pre: *b* ∈ $\mathcal{B}$, *e* ∈ *TElem*
  - post: *b'*∈ $\mathcal{B}$, *b'*=*b* \ {e} (ein Vorkommen/occurrence des Elementes *e* wurde aus dem Bag entfernt). Falls *e* nicht in *b* enthalten war, dann bleibt *b* unverändert

$$\text{remove} \leftarrow \begin{cases} true, falls\ ein\ Element\ entfernt\ wurde (size(b') < size(b)) \\ false, falls\ e\ nicht\ in\ b\ enthalten\ war\ (size(b') = size(b)) \end{cases}$$

ADT Bag - Interface

- search(b, e)
  - pre: $b \in \mathcal{B}$, $e \in TElem$
  - post: search $\leftarrow \begin{cases} wahr, & falls\ e \in b \\ falsch, & ansonsten \end{cases}$
- size(b)
  - pre: $b \in \mathcal{B}$
  - post: size ← Anzahl der Elemente aus b
- nrOccurrences(b,e)
  - pre: $b \in \mathcal{B}$, $e \in TElem$
  - post: nrOccurences ← Vorkommen des Elementes e in b

ADT Bag - Interface

- isEmpty(b)
  - descr**:** überprüft ob das Bag leer ist
  - pre: $b \in \mathcal{B}$
  - post: *isEmpty* $\leftarrow \begin{cases} wahr, & falls\ b\ keine\ Elemente\ enthält \\ falsch, & ansonsten \end{cases}$
- destroy(b)
  - pre: $b \in \mathcal{B}$
  - post: *b* wurde zerstört
- iterator(b, i)
  - pre: $b \in \mathcal{B}$
  - post: $i \in \mathcal{I}$, *i* ist ein Iterator für *b*

ADT Bag – Repräsentierung A

- **Ein dynamisches Array, wo die Elemente vielfach vorkommen können**
- Zum Beispiel, wen das Bag folgende Elemente enthält: 1, 3, 2, 6, 2, 5, 2, dann kann man diese in einem Array folgendermaßen speichern:

1	3	2	6	2	5

- Lösche das Element 6:

1	3	2	5	2	

ADT Bag – Repräsentierung B

- **Ein dynamisches Array von Paaren der Form (*Element, Frequenz*),** wobei die Elemente eindeutig sind, und zusätzlich speichert man für jedes Element die Frequenz
- Zum Beispiel, wen das Bag folgende Elemente enthält: 1, 2, 5, 2, 3, 3, 1, 2, 1, 2, 6 dann kann man diese in einem Array folgendermaßen speichern:

(1, 3)	**(2, 4)**	**(5, 1)**	**(3, 2)**	**(6, 1)**		

- oder

elems	**1**	**2**	**5**	**3**	**6**		
freq	**3**	**4**	**1**	**2**	**1**		

ADT Bag – Repräsentierung B

elems	**1**	**2**	**5**	**3**	**6**		
freq	**3**	**4**	**1**	**2**	**1**		

- Füge das Element 2 ein:

elems	**1**	**2**	**5**	**3**	**6**		
freq	**3**	**5**	**1**	**2**	**1**		

- Füge das Element -5 ein:

elems	**1**	**2**	**5**	**3**	**6**	**-5**	
freq	**3**	**5**	**1**	**2**	**1**	**1**	

ADT Bag – Repräsentierung B

elems	**1**	**2**	**5**	**3**	**6**	**-5**	
freq	**3**	**5**	**1**	**2**	**1**	**1**	

- Lösche das Element 5:

elems	**1**	**2**	**-5**	**3**	**6**		
freq	**3**	**4**	**1**	**2**	**1**		

- Lösche das Element 2:

elems	**1**	**2**	**-5**	**3**	**6**		
freq	**3**	**3**	**1**	**2**	**1**		

ADT Bag – Repräsentierung C

- **Ein dynamisches Array von Frequenzen** aufgebaut folgendermaßen:
  - Man berechnet das aktuelle Werte Intervall [*a,b*] (funktioniert nur wenn die Werte ganze Zahlen sind)
  - Das Werte Intervall wird in das Intervall [1,*x*] übersetzt, sodas $x = b - a+1$
  - In dem Array speichert man:
    - Auf Position 1 die Frequenz des Wertes *a* (das minimale Wert)
    - Auf Position 2 die Frequenz des Wertes *a+1*
    - …
    - Auf Position *x-1* die Frequenz des Wertes *b-1*
    - Auf Position *x* die Frequenz des Wertes *b*
- Wenn ein Wert eingefügt wird, welcher sich außerhalb des Intervalls [*a,b*] befindet, dann muss das Werteintervall und folglich auch das Array von Frequenzen aktualisiert werden

ADT Bag – Repräsentierung C

- Nehmen wir als Beispiel ein Bag, das folgende Elemente enthält:

  5, 10, -1, 2, 3, 10, 5, 5, -5

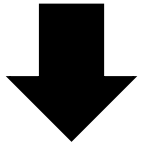
- Das Werteintervall [-5,10] wird in das Intervall [1,16] übersetzt, i.e. :
  - auf der Position 1 speichert man die Frequenz des Wertes -5 (minimale Wert), auf der Position 2 die Frequenz des Wertes -4, …, auf der Position 16 die Frequenz des Wertes 10
- Es wird in dem Array folgendermaßen gespeichert:

1	**0**	**0**	**0**	**1**	**0**	**0**	**1**	**1**	**0**	**3**	**0**	**0**	**0**	**0**	**2**
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

ADT Bag – Repräsentierung C

- Wenn man das Element -7 einfügt, dann ändert sich das Werteintervall: [-7,10]
- Das heißt Position 1 in dem Array entspricht jetzt dem Wert -7 (alle Werte müssen mit zwei Positionen nach rechts verschoben werden)
- Bemerkung: wenn *min* das minimale Wert ist, dann ist die **Position** eines Elementes ***e*** genau ***e-min+1***

1	0	0	0	1	0	0	1	1	0	3	0	0	0	0	2



1	0	1	0	0	0	1	0	0	1	1	0	3	0	0	0	0	2

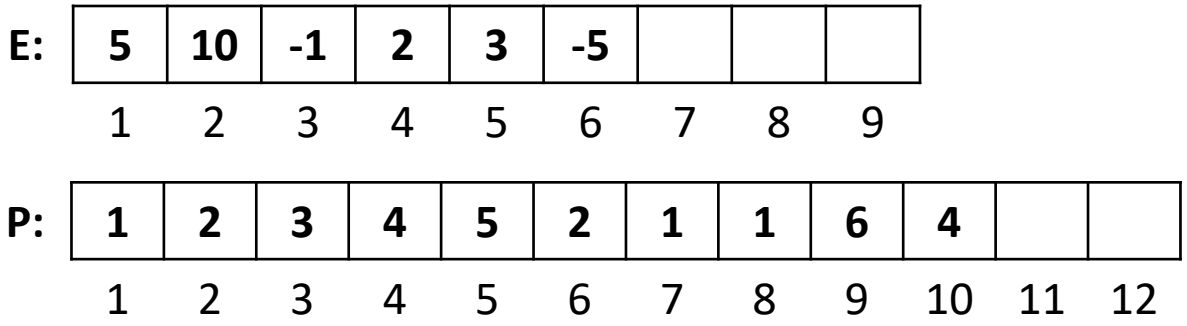
ADT Bag – Repräsentierung D

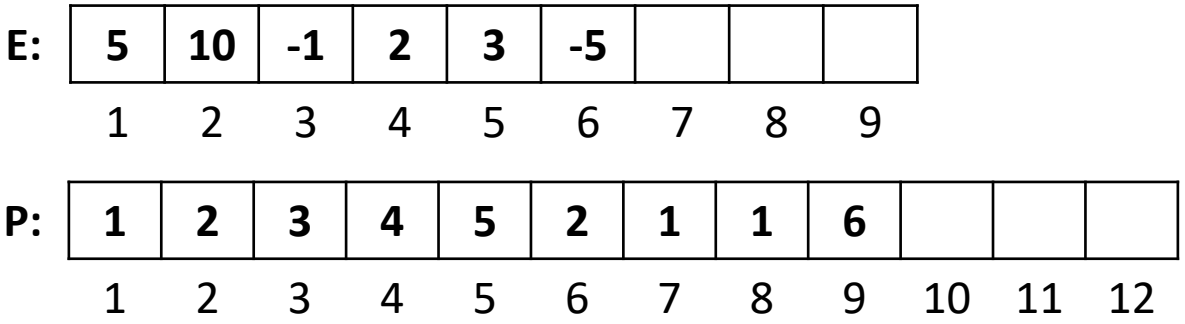
- Ein dynamisches Array von eindeutigen Elementen (E) und ein dynamisches Array von Positionen (P) in E für die Elemente des Bags
- Das Array E enthält also die Menge aller Werte des Bags
- Das Array P hat folgende Bedeutung:
  - P[1] = a, i.e. das erste Element des Bags ist E[a]
  - P[2] = b, i.e. das nächste Element des Bags ist E[b]
  - ...
- Nehmen wir als Beispiel ein Bag, das folgende Elemente enthält:

  5, 10, -1, 2, 3, 10, 5, 5, -5
- Es wird folgendermaßen repräsentiert:
  - E = [5, 10, -1, 2, 3, -5]
  - P = [1, 2, 3, 4, 5, 2, 1, 1, 6]

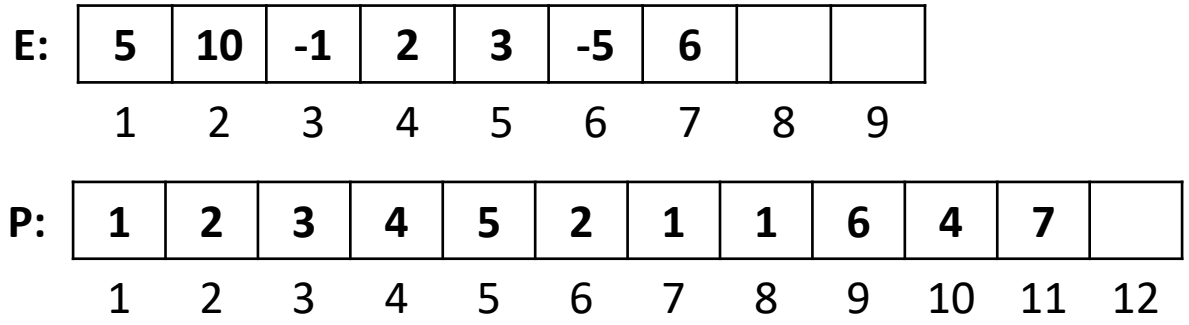
ADT Bag – Repräsentierung D

- Füge das Element 2 ein:

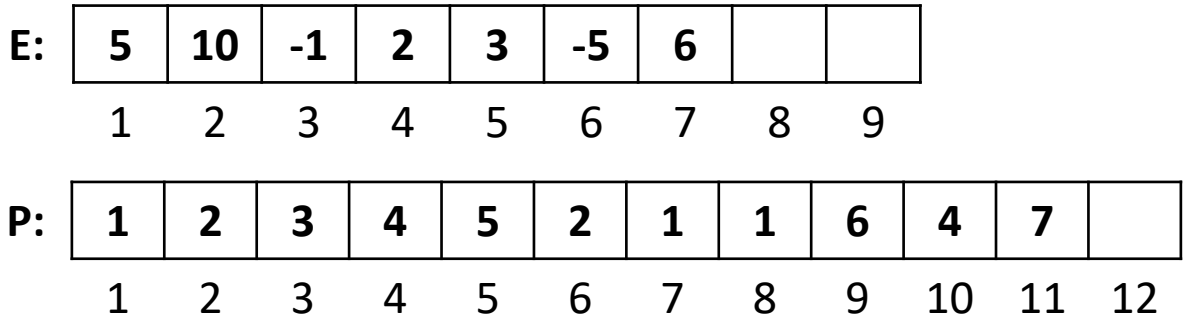




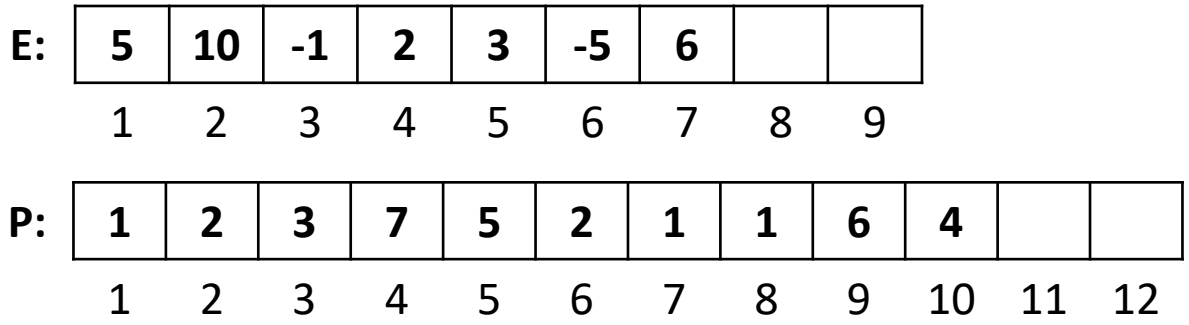
- Füge das Element 6 ein:



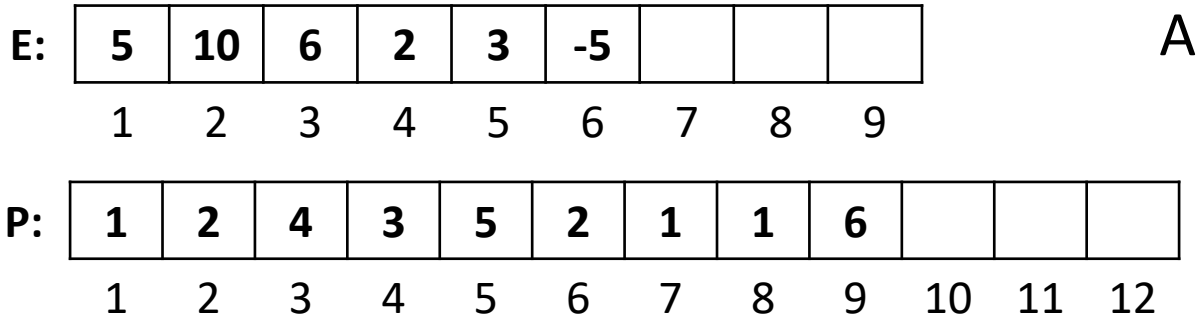
ADT Bag – Repräsentierung D



- Lösche das Element 2:



- Lösche das Element -1:



Aufpassen!

ADT Bag – Repräsentierung

- Bemerkung. Für Repräsentierung A und B können auch andere Datenstrukturen benutzt werden.
- Repräsentierung C und D sind spezifisch für ein dynamisches Array.

ADT Sorted Bag

- In einem Bag können die Elemente basierend auf einer Ordnungsrelation sortiert werden → *SortedBag*

- In diesem Fall enthält das SortedSet Elemente vom Typ *TComp* anstatt Elemente vom Typ *TElem*

ADT Sorted Bag

- Welche Operationen muss man ändern?
- Die einzigen Änderungen zu dem Interface sind bei der *init* Operation, wo man auch die Relation als Parameter hat

- Domäne: $S\mathcal{B}$ = {sb | sb ist ein Sorted Bag mit Elementen vom Typ TComp}
- init(sb, rel)
  - **desc:** erstellt einen leeren Sorted Bag, wo die Elemente basierend auf der Relation *rel* geordnet werden
  - **pre:** *rel* ∈ Relation
  - **post:** *sb* ∈ $S\mathcal{B}$, *sb* ist ein leeres SortedBag mit *rel* als Relation

ADT Sorted Bag

- Die Relation bestimmt wie die Elemente sortiert werden (z.B. steigende Reihenfolge, alphabetische Reihenfolge, usw.)
- Die Relation wird als Funktion mit zwei Parametern definiert (die zwei Elemente, die verglichen werden) , welche *true* zurückgibt, falls die Elemente in der richtigen Reihenfolge sind, oder *false,* falls die Elemente nicht in der richtigen Reihenfolge sind.

ADT Sorted Bag - Iterator

- Für einen sortierten Bag muss der Iterator die Elemente in der Reihenfolge gegeben von der Relation durchlaufen

- Damit die Iterator Operationen Komplexität $\Theta(1)$ haben, sollen die Elemente sortiert gespeichert werden

Bag/SortedBag - Repräsentierung

- Um ADT Bag (oder ADT SortedBag) zu implementieren kann man folgende Datenstrukturen für die Repräsentierung benutzen:
  - (dynamisches) Array
  - Verkettete Liste
  - Hashtabellen
  - (balancierte) Binärbäume – für sortierte Sets
  - Skip Listen – für sortierte Bags

Bag/SortedBag - Repräsentierung

- Unabhängig von der Datenstruktur gibt es zwei Möglichkeiten, die Elemente zu speichern:
  - Alle Elemente separat zu speichern (in der entsprechenden Reihenfolge)
  - Man speichert eindeutige Elemente (in der entsprechenden Reihenfolge) und die Frequenzen

Aufgabe

- Um Wahlbetrug zu vermeiden (jede Person darf ein einziges Mal wählen) braucht man eine Anwendung um die Personalnummer der Personen zu speichern, die schon gewählt haben.
- Welche Eigenschaften sollte der Container haben?
  - Die Elemente sollen eindeutig sein
  - Die Reihenfolge der Elemente ist nicht wichtig

- Ein Container, deren Elemente eindeutig sind, wobei die Reihenfolge der Elemente keine Rolle spielt (die Elemente haben keine entsprechende Positionen) ist **ADT Set**

ADT Set

- Es gibt keine Operationen basierend auf Positionen
- Die Elemente sind nicht unbedingt in derselben Reihenfolge gespeichert, in der sie eingefügt wurden

- Domäne für ADT Set:

  $\mathcal{S}$ = {s | s ist ein Set mit Elementen vom Typ TElem}

Set – Interface

- init(s)
  - **descr:** erstellt einen leeren Set
  - **pre:** wahr
  - **post**: *s* ∈ $\mathcal{S}$, *s* ist einen leeren Set

- add(s, e)
  - **descr**: fügt ein neues Element zu dem Set ein
  - **pre**: *s* ∈ $\mathcal{S}$, *e* ∈ *TElem*
  - **post**: *s'* ∈ $\mathcal{S}$, *s'* = *s* ∪ {e} (TElem *e* wird in dem Set eingefügt nur falls er noch nicht in dem Set enthalten war, ansonsten bleibt *s* unverändert)

$$\text{add} \leftarrow \begin{cases} true, falls\ ein\ Element\ eingefügt\ wurde (size(s') > size(s)) \\ false, falls\ e\ schon\ in\ b\ enthalten\ war\ (size(s') = size(s)) \end{cases}$$

Set – Interface

- remove(s, e)
  - **descr**: löscht ein Element aus dem Set
  - **pre**: *s* ∈ $\mathcal{S}$, *e* ∈ *TElem*
  - **post**: *s'* ∈ $\mathcal{S}$, *s'* = *s* \ {e} (Falls *e* nicht in *s* enthalten war, dann bleibt *s* unverändert)

$$\text{remove} \leftarrow \begin{cases} true, falls\ ein\ Element\ entfernt\ wurde (size(s') < size(s)) \\ false, fall\ e\ nicht\ in\ b\ enthalten\ war\ (size(s') = size(s)) \end{cases}$$

- search(s, e)
  - **descr**: sucht ob ein Element in dem Set enthalten ist
  - **pre**: *s* ∈ $\mathcal{S}$, *e* ∈ *TElem*
  - **post**: search $\leftarrow \begin{cases} wahr, & falls\ e \in s \\ falsch, & ansonsten \end{cases}$

Set – Interface

- size(s)
  - **descr**: gibt die Anzahl der Element aus dem Set zurück
  - **pre**: $s \in S$
  - **post**: size ← Anzahl der Elemente aus *s*

- isEmpty(s)
  - **descr:** überprüft ob das Set leer ist
  - **pre**: $s \in S$
  - **post**: *isEmpty* $\leftarrow \begin{cases} wahr, & falls\ s\ keine\ Elemente\ enthält \\ falsch, & ansonsten \end{cases}$

Set – Interface

- iterator(s, i)
  - **descr**: gibt ein Iterator für ein Set zurück
  - **pre**: $s \in S$
  - **post**: $i \in I$, i ist ein Iterator für s

- destroy(s)
  - **descr**: zerstört ein Set
  - **pre**: $s \in S$
  - **post**: s wurde zerstört

Set – Interface

- Andere mögliche Operationen (spezifisch für mathematische Mengen):
  - Vereinigung zweier Mengen
  - Durchschnitt zweier Mengen
  - Differenz zweier Mengen

SortedSet

- In einem Set können die Elemente basierend auf einer Ordnungsrelation sortiert werden → *SortedSet*
- In diesem Fall enthält das SortedSet Elemente vom Typ *TComp* anstatt Elemente vom Typ *TElem*
- Die einzigen Änderungen zu dem Interface sind bei der *init* Operation, wo man auch die Relation als Parameter hat
- Für einen sortierten Set muss der Iterator die Elemente in der Reihenfolge gegeben von der Relation durchlaufen

Set/SortedSet - Repräsentierung

- Um ADT Set (oder ADT SortedSet) zu implementieren kann man folgende Datenstrukturen für die Repräsentierung benutzen:
  - (dynamisches) Array:
    - Array von Elementen oder
    - Bitarrays
  - Verkettete Liste
  - Hashtabellen
  - (balancierte) Binärbäume – für sortierte Sets
  - Skip Listen – für sortierte Sets

ADT Set – Repräsentierung A

- **Ein dynamisches Array, wo die Elemente eindeutig sind**
- Zum Beispiel, wen das Set folgende Elemente enthält: 1, 3, 2, 6, 5 dann kann man diese in einem Array folgendermaßen speichern:

1	3	2	6	5

ADT Set – Repräsentierung B

- **Ein dynamisches Array von Boolean Werten (Bitarray)**
  - Man berechnet das aktuelle Werte Intervall [*a,b*]
  - Das Werte Intervall wird in das Intervall [1,*x*] übersetzt, sodas *x= b – a+1*
  - In dem Array speichert man:
    - Auf Position 1 True, falls der Wert *a* (das minimale Wert) zu dem Set gehört, False ansonsten
    - Auf Position 2 True, falls der Wert *a+1* zu dem Set gehört, False ansonsten
    - ...
    - Auf Position *x-1* True, falls der Wert *b-1* zu dem Set gehört, False ansonsten
    - Auf Position *x* True, falls der Wert *b* zu dem Set gehört, False ansonsten
- Wenn ein Wert eingefügt wird, welcher sich außerhalb des Intervalls [*a,b*] befindet, dann muss das Werteintervall und folglich auch das Bitarray aktualisiert werden

ADT Set – Repräsentierung B

- Nehmen wir als Beispiel ein Set, das folgende Elemente enthält:

  5, 10, -1, 2, 3, -5

- Das Werteintervall [-5,10] wird in das Intervall [1,16] übersetzt, i.e. :
  - auf der Position 1 speichert man ob der Wert -5 (minimale Wert) zu dem Set gehört, auf der Position 2 ob der Wert -4 zu dem Set gehört, …, auf der Position 16 ob der Wert 10 zu dem Set gehört
- Es wird in einem Boolean Array folgendermaßen gespeichert:

True	False	False	False	True	False	False	True	True	False	True	False	False	False	False	True

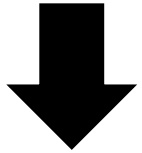
- Oder in einem Bitarray:

1	0	0	0	1	0	0	1	1	0	1	0	0	0	0	1

ADT Set – Repräsentierung B

- Wenn man das Element -7 einfügt, dann ändert sich das Werteintervall: [-7,10]
- Das heißt Position 1 in dem Array entspricht jetzt dem Wert -7 (alle Werte müssen mit zwei Positionen nach rechts verschoben werden)
- Das Bitarray muss folgendermaßen geändert werden:

1	0	0	0	1	0	0	1	1	0	1	0	0	0	0	1



1	0	1	0	0	0	1	0	0	1	1	0	1	0	0	0	0	1