

Ordbøker

I. ORDBOK / PRIORITETSKØ = SEKVENNS / LIFI

vilkårlig / minste nøkkel

vilkårlig / første Posisjon

II. IMPLEMENTASJON MED SEQUENCE

III. IMPLEMENTASJON MED HASH TABELL

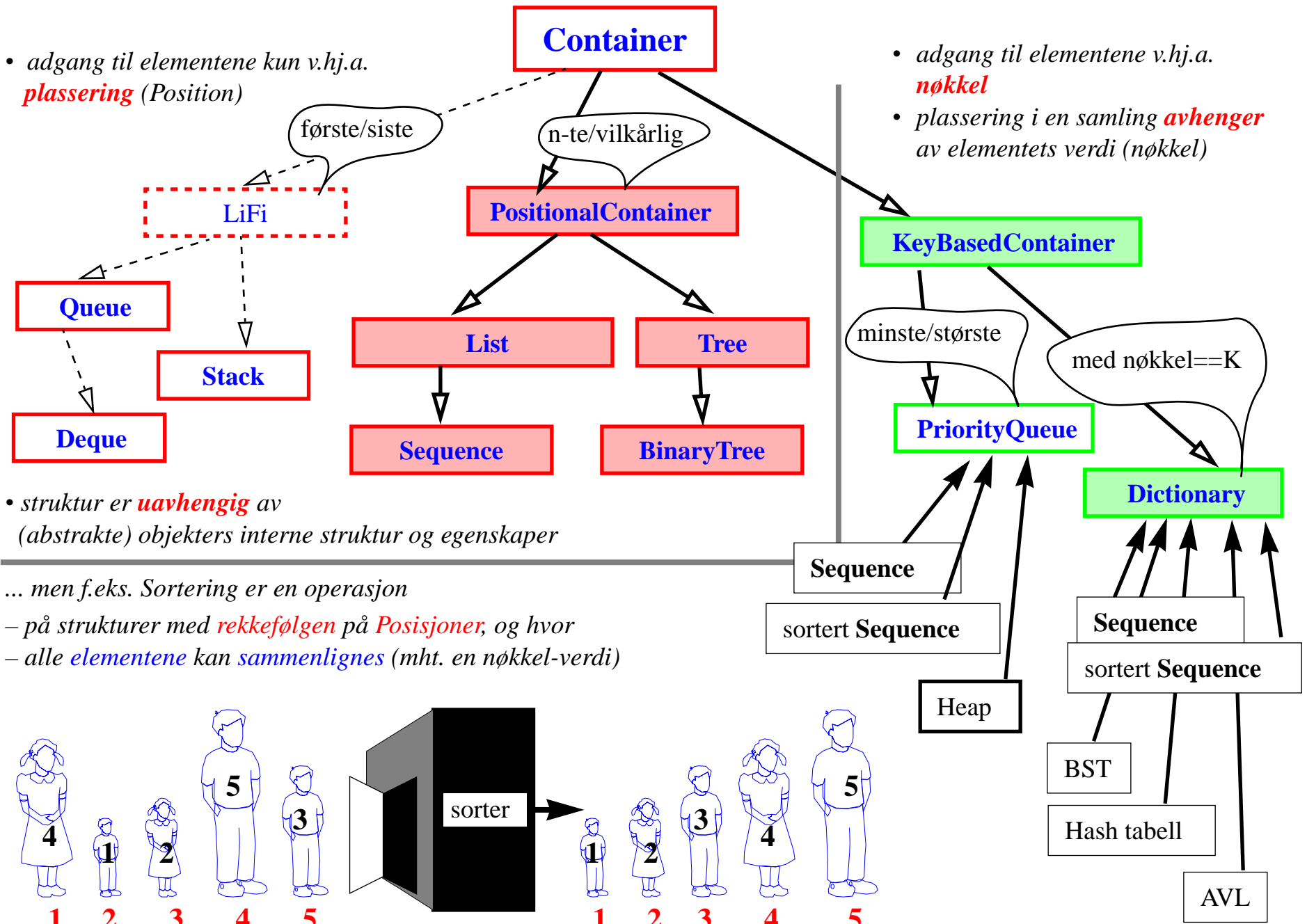
hash funksjoner

håndtering av kollisjoner

IV. IMPLEMENTASJON MED BST (BINARY SEARCH TREE)

Kap. 8: (kursorisk: 8.3.3 – 8.3.7, 8.7;
unntatt: 8.6)

Kap. 9: (unntatt: 9.2.1 – 9.7)



En Ordbok

er en funksjon

f: Nøkler → *DataObjekter*

personnummer → Person
fødselsdato + kjønn (+???) → personnummer
navn + adresse → telefonnummer
navn → telefonnummer
studentnavn (+???) → studentID
studentID → karakterutskrift
x → x^2

som tillatter en

- å sette inn nye nøkkel-objekt par
- finne et objekt med en gitt nøkkel
- (finne alle objektene med en gitt nøkkel)
- fjerne et objekt med en gitt nøkkel
- finne neste/forrige objektet (mht. nøkkel)

```
public interface DictionaryWithoutLocators extends KeyBasedContainer {  
    void insert (Object key, Object o)  
    Object find (Object key)  
    ObjectIterator findAll (Object key)  
    Object remove (Object key)  
    -----  
    Object closestElemAfter (Object key)  
    Object closestElemBefore (Object key)  
    Object closestKeyAfter (Object key)  
    Object closestKeyBefore (Object key)  
}
```

Dictionary med Locator

```
public interface (Ordered)DictionaryWithoutLocators ... {  
    void insert                (Object key, Object o)  
    Object remove              (Object key)  
    Object find                (Object key)  
    ObjectIterator findAll     (Object key)  
    Object closestElemAfter    (Object key)  
    Object closestElemBefore  (Object key)  
    Object closestKeyAfter     (Object key)  
    Object closestKeyBefore   (Object key)  
}
```

```
public interface Locator {  
    Object element()  
    Object key()  
}
```

```
public interface (Ordered)Dictionary ... {  
    Locator insert            (Object key, Object o)  
    void remove              (Locator loc)  
    Locator find            (Object key)  
    LocatorIterator findAll (Object key)  
    Locator after           (Locator loc)  
    Locator before          (Locator loc)  
    Locator closestAfter    (Object key)  
    Locator closestBefore  (Object key)  
}
```

Dictionary og OrderedDictionary

```
public interface KeyBasedContainer extends Container {  
    LocatorIterator locators()  
    ObjectIterator keys()  
    /** Inserts a <key, element> pair */  
    Locator insert(Object k, Object o)  
    /** Removes an element from this Container. */  
    void remove(Locator)  
    /** Inserts a Locator into this Container. */  
    void insert(Locator)  
    Object replaceElement(Locator l, Object o)  
    /** Changes the key of Locator's element.  
    @return the old key of this Locator's element */  
    Object replaceKey(Locator, Object)  
}
```

package jdsl.core.api

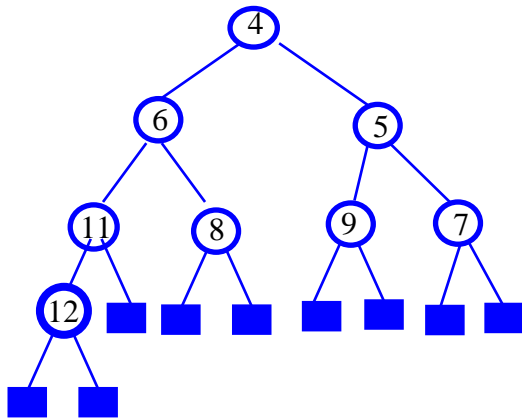
```
public interface Locator {  
    Object element()  
    Object key()  
}
```

```
public interface Dictionary extends KeyBasedContainer {  
    /** @param key to search for an object  
    @returns Locator mapped to key – NO_SUCH_KEY if not found  
    @exception InvalidKeyExc if key is not valid for this Cotainer */  
    Locator find(Object key)  
    /** @param key to search for an object  
    @returns all Locators mapped to key – empty if not found  
    @exception InvalidKeyExc if key is not valid for this Cotainer */  
    LocatorIterator findAll(Object key)  
    /** Special value (rather than exception) returned when no object with  
    a specified key was found */  
    Locator NO_SUCH_KEY;
```

```
public interface OrderedDictionary extends Dictionary {  
    /** @returns Locator sequentially after/before loc – BOUNDARY_VIOLATION if goes out of scope  
    @exception InvalidLocatorExc if loc is invalid (e.g. not within this Container) */  
    Locator after/before(Locator loc)  
    /** @returns a Locator whose key is equal to or just after/before key – BOUNDARY_VIOLATION if no such exists  
    @exception InvalidKeyExc if key is invalid for this Container */  
    Locator closestAfter/Before(Object key)  
    /** Special Locator (rather than exception) returned when a requested Locator was not found */  
    Locator BOUNDARY_VIOLATION;
```

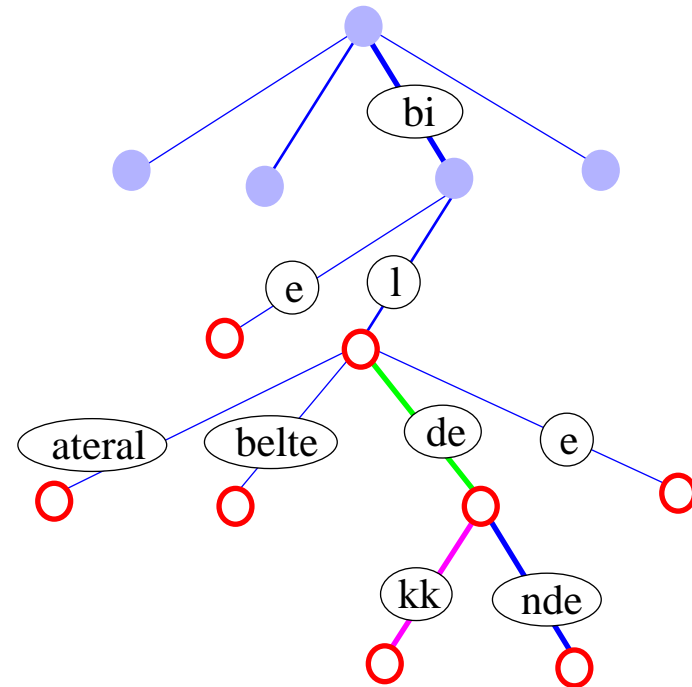
Implementasjon av Dictionary

ikke Haug



hvordan finner man '11' ... ?

ikke Tries



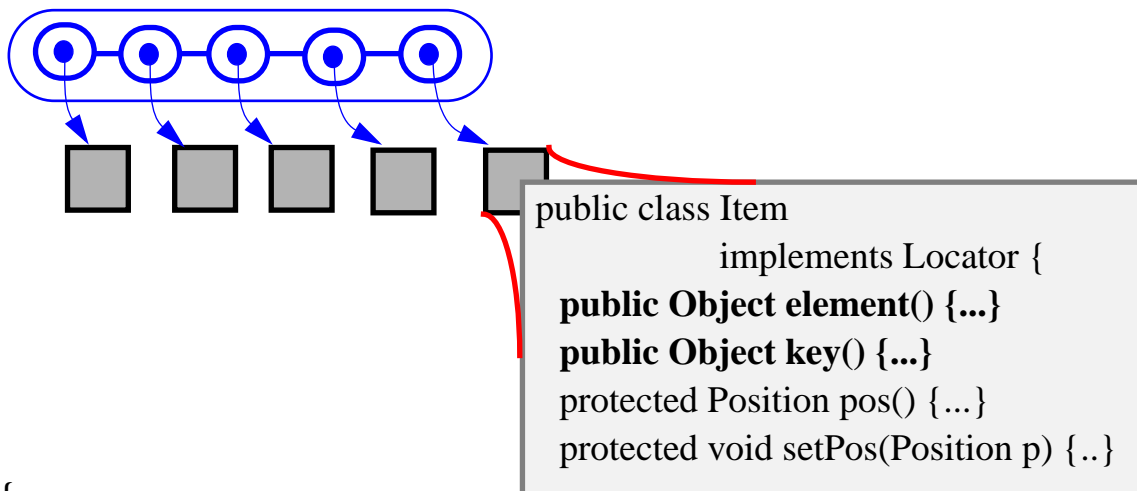
hva om nøkler ikke er String/Sequence ... ?

II. Implementasjon av Dictionary – *med Sequence*

I DATA REPRESENTASJON

Sequence, der hver Position

lagrer en Item



II DATA STRUKTUR

```
class DicSequence implements Dictionary {
```

```
    private Sequence S;
```

```
    private Comparator cp;
```

```
    /** @param sq bør være tom sekvens
```

```
        @param c Comparator for sammenlikning av Item med hensyn til key-objekter */
```

```
    public DicSequence(Sequence sq, Comparator c) {
```

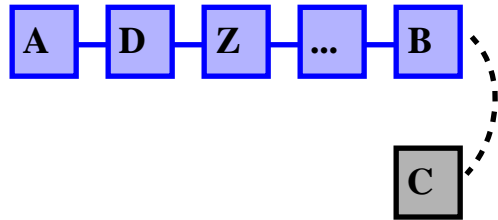
```
        S = sq; cp = c; }
```

III DATA INVARIANT

- Item i S kan sammenlignes med cp
- S er usortert / S er sortert

Implementasjon av Dictionary – med Sequence (med LL/DL)

DATA INVARIANT: INGEN – USORTERT



void insert (k,o) O(1)
 sq.insertLast (new Item(k,o));

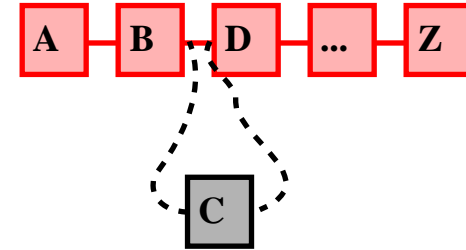
Object find (Object k): O(n)
 Item ko = new Item(k,null);
 Object ret; boolean fant = false;
 Position p = sq.first();
 try{ while (!fant) {
 if (cp.isEqualTo(p.element(), ko)
 { ret = ((Item)p.element()).element();
 fant = true; }
 else p = sq.after(p);
 } }
 catch (BoundaryExc e) { ret = null;}
 return ret;

Object remove (Object k) : find ... O(n)

ObjectIterator findAll (Object k) : find ... Θ(n)

Object closestAfter/Before (Object k) : find ... Θ(n)

DATA INVARIANT: SORTERT STIGENDE



void insert (k,o) O(n)

```
Item ny= new Item(k,o);
if ( sq.isEmpty() )
    sq.insertFirst(ny)
else if ( cp.isLessOrEqual(ny, sq.first().element() ) )
    sq.insertFirst(ny)
else if ( cp.isGreaterOrEqual(ny, sq.last().element() ) )
    sq.insertLast(ny)
else Position c = sq.first()
    while (cp.isGreaterThan(ny, c.element() )
        c= sq.after(c)
    sq.insertBefore(c,ny)
```

Object find (Object k) : ... O(n)

Object remove (Object k) : find ... O(n)

ObjectIterator findAll (Object k) : find ... O(n)

Object closestAfter/Before (Object k) : find ... O(n)

Implementasjon av Dictionary – med Sequence (med Array)

III. DATA INVARIANT :

Sortert Array A[l..h] :

- hvis $i < j$ så **cp.isLessOrEqual**(A[i], A[j])
- hvis **cp.isLessThan**(A[i], A[j]) så $i < j$
- hvis **cp.isGreaterThan**(A[i], A[j]) så $i > j$

< **cp.isLessThan**

[0]	11	← l = 0
[1]	19	
[2]	24	
[3]	32	
[4]	32	← m = (0+9)/2
[5]	48	
[6]	50	
[7]	55	
[8]	72	
[9]	99	← h = 9

[0]	11	
[1]	19	
[2]	24	
[3]	32	
[4]	32	
[5]	48	← l = 5
[6]	50	
[7]	55	← m = (5+9)/2
[8]	72	
[9]	99	← h = 9

[0]	11	
[1]	19	
[2]	24	
[3]	32	
[4]	32	
[5]	48	← l = 5
[6]	50	← h = 6
[7]	55	
[8]	72	
[9]	99	

$m = (5+6)/2$
A[m] == 48

find(k) = BinærSøk(k) = O(log n)

BinærSøk

```
find(k) : find(k, l, h)
if ( l > h ) return -1
else
  m = (l+h) / 2
  if ( cp.isEqualTo(A[m],k) )
    return A[m].element();
  else if ( cp.isLessThan(A[m],k) )
    return finn(k,m+1,h)
  else // cp.isGreaterThan(A[m],k)
    return finn(k,l,m-1)
```

Implementasjon av Dictionary (så langt)

operasjon	usortert Sequence		sortert Sequence	
	Array	DL	Array	DL
Container				
size	1	1	1	1
isEmpty	1	1	1	1
elements	n	n	n	n
Dictionary				
find	n	n	log n	n
findAll	n	n	log n (+ s)	n
insert	1	1	n	n
remove	n	n	n	n
closestAfter/Before	n	n	n	n

III. Implementasjon av Dictionary – med HashTabell

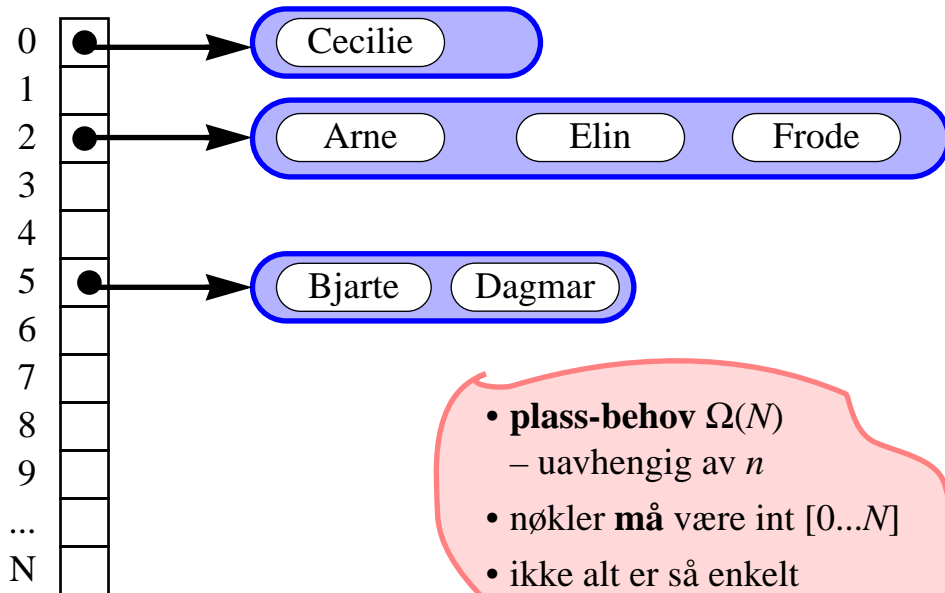
key : elementer	→ nøkkel [heltall]	injektiv	TO
Personer	→ personnummer	+	+
Personer	→ fødselsår	-	+
variable i et program	→ type (int, boolean, Tree...)	-	-

Arne(1972), Bjarte(1975), Cecilie(1970), Dagmar(1975), Elin(1972), Frode(1972)

```
int key(Person p) { return p.fødselsår }
int hash(int k) { return k - 1970 }
```

key : elementer → nøkkel
 hash : int nøkkel → int hashkode

Sequence[] H = new Sequence[N]



- plass-behov $\Omega(N)$
– uavhengig av n
- nøkkel må være int $[0...N]$
- ikke alt er så enkelt

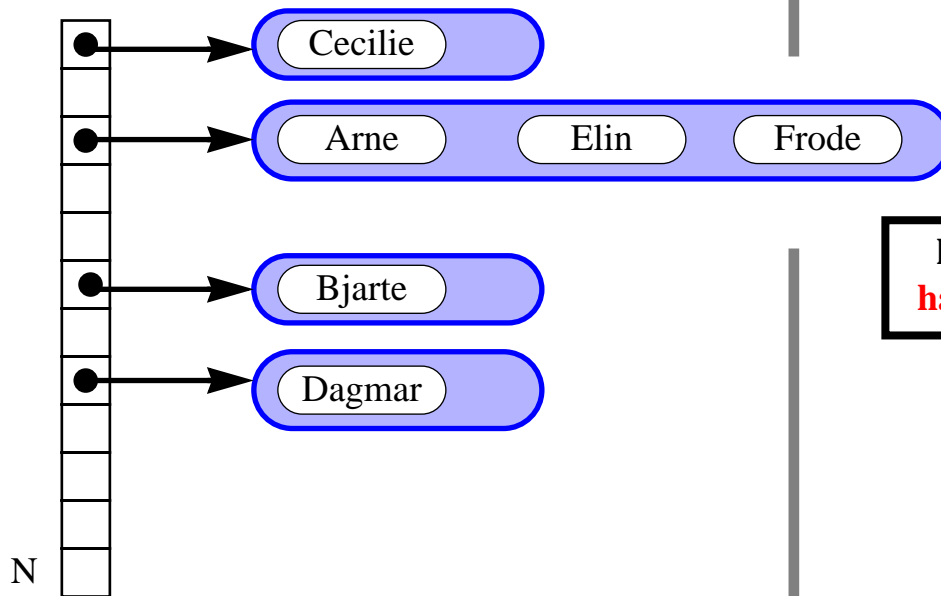
```
insert(int k, Object o) { O(1)
    H[hash(k)].insertFirst(o) }

find(int k) { O(1)
    Sequence S = H[hash(k)]
    return S.first().element() }

remove(int k) { O(1)
    Sequence S = H[hash(k)]
    S.remove(S.first()) }

remove(Person p) { O(|H[hash(k)]|)
    k = p.fødselsår;
    Sequence S = H[hash(k)]
    finn p i S }
```

Sequence[] H = new Sequence[N]



```

int hash(int k) { return k - 1970 }

insert(int k, Object o) {
    H[hash(k)].insertFirst(o) }           O(1)

find(int k) {
    Sequence S = H[hash(k)]
    return S.first().elem() }             O(1)

remove(int k) {
    Sequence S = H[hash(k)]
    S.remove(S.first()) }                  O(1)
    
```

injektiv hash:
 hash(k1) = hash(k2) hviss k1 = k2

Et typisk eksempel ...

Nøkkel er personnummer og vi skal samle alle ansatte.
 Vi kan ikke indeksere tabellen med persnr
 – det er jo bare 250 ansatte!!

key : personer → persnr
 hash : int persnr → int siste 4 sifrene

Arne	123456 12345	2345
Elin	654321 22345	2345
Frode	212131 92345	2345

key er injektiv men hash er det ikke

```

insert(int pn, Object o) {
    H[hash(pn)].insertFirst(o) }           O(1)

find(int pn) {
    Sequence S = H[hash(pn)]
    finn pn i S }                           O(|S|)

remove(int pn) {
    Sequence S = H[hash(pn)]
    finn pn i S og fjern }                   O(|S|)
    
```

GENERELT: hash er ikke injektiv
 forventet/gjennomsnittlig |S| = load factor = n/N < 1

Hash funksjoner

1. Gitt en nøkkel n – et Objekt! – konverter den til int $k = k(n)$
2. Gitt en int-representasjon av nøkkel $k = k(n)$ – finn en hashkode $hash(k)$
 - a) hvis `cp.isEqualTo(n1,n2)` så $hash(k(n1)) == hash(k(n2))$ der `cp` er den aktuelle **Comparator** for nøkler
 - b) $hash(k) < N$ – der N er aktuell størrelse av tabellen
 - c) $hash$ skal gi “jevn distribusjon” – unngå kollisjoner

1. **Key** bør men trenger ikke å være injektiv:

$k : \text{String} \rightarrow \text{int}$

$k(\text{streng}) = \text{summen av ASCII koder av alle tegn i } \text{streng}'\text{en}$

$k(\text{"alle"}) = 97 + 108 + 108 + 101 = 414$

$k(\text{"anna"}) = 97 + 110 + 110 + 97 = 414$

A	– 65
...	
a	– 97
e	– 101
l	– 108
n	– 110

2. Hash

- a) hvis $k1 == k2$, så $hash(k1) == hash(k2)$, siden $hash$ er en funksjon
- b) for hver k er : $hash(k) = k \bmod N < N$
- c) ingen “beste, generelle” hash-funksjon :
 - unngå konflikter : $hash(k) = 1$ er ikke bra
 - minimaliser gjennomsnittlig lengde for hver samling $hash(k)$: $find(k) = \text{finn } k \text{ i Container } hash(k)$
 - alt avhenger av forventet distribusjon av data-nøkler ...

– N bør være et primtall

20	30	40	mod 10 :	0	0	0	mod 11 :	9	8	7
23	33	43		3	3	3		1	0	10
25	35	45		5	5	5		3	2	1

– ofte brukt hash-funksjon

$hash(k) = (a * k + b) \bmod N$, der N er et primtall, $a > 0$, $b \geq 0$

Linear Probing

Ansatte identifiseres med
nøkkel = personnr.: 11-siffer heltall

ansatt[] H[98765432101]

men det er kun 250 ansatte.

ansatt[] H[250]

hash:12345676275

mod 250 ...

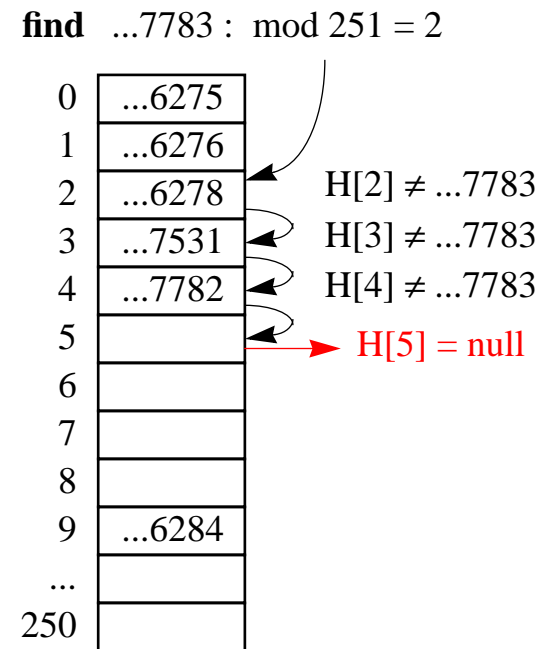
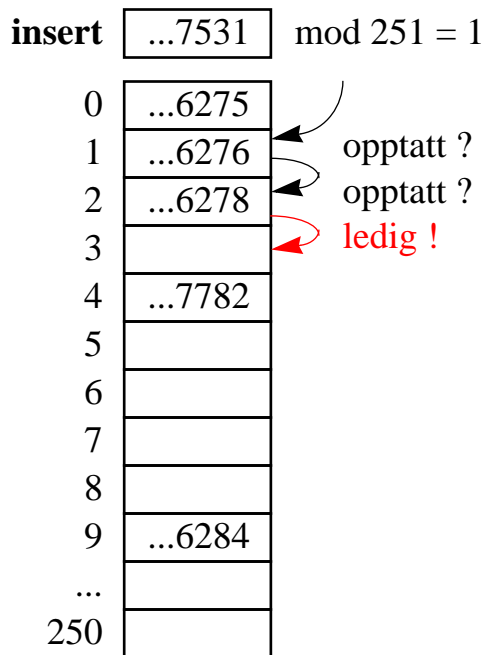
mod 251 ...

siste 4 siffre mod 251

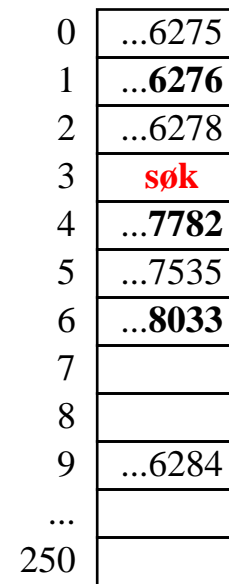
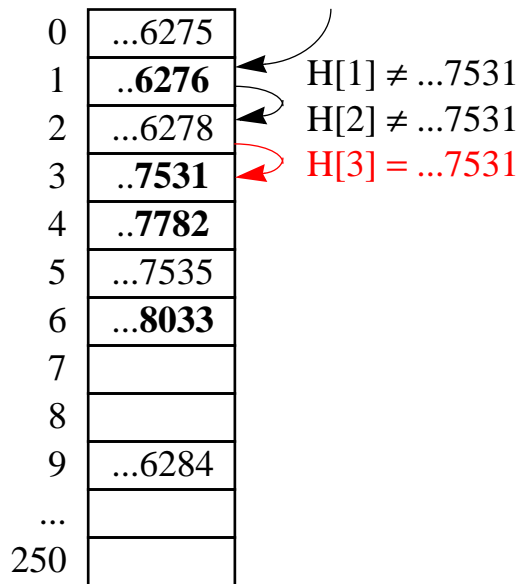
hash(12345676275)
= 6275 mod 251 = 0

ansatt[] **H[251]**

```
int hash(int k) {
    s = siste 4 siffrene fra k
    return (s % 251) }
```

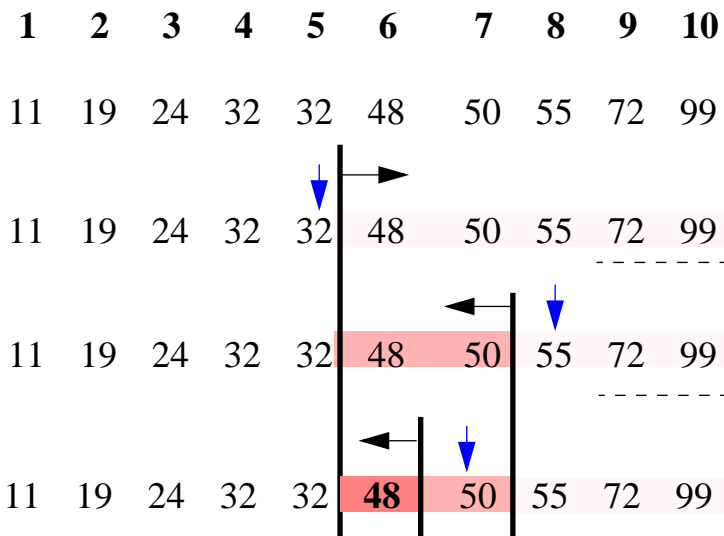


remove ...7531 : mod 251 = 1



IV. Binært Søk → Binære Søketrær

find(48)



for alle m, x :

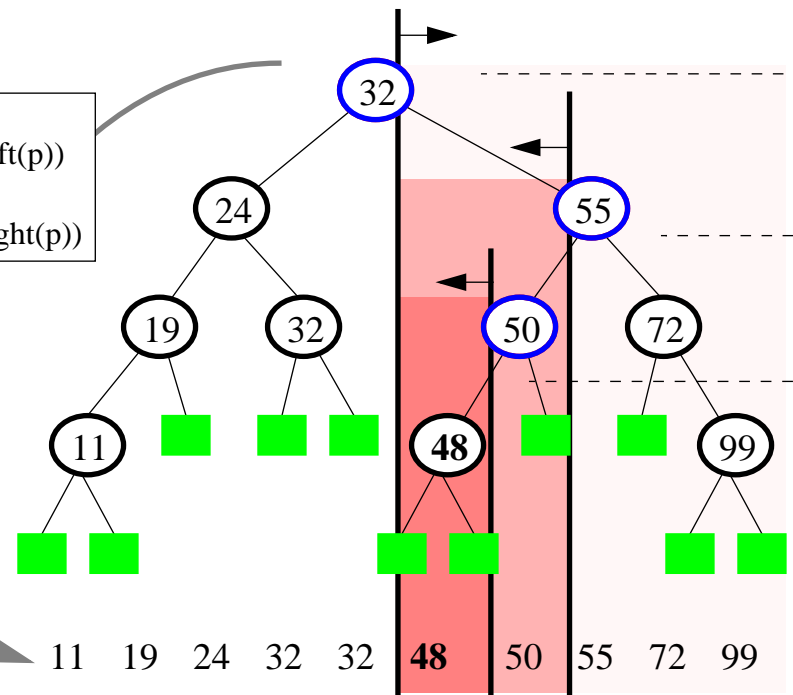
- $A[x] > A[m] : x > m$
- $A[x] < A[m] : x < m$

: $A[m]=32 < 48$ – søk til høyre

: $A[m]=55 > 48$ – søk til venstre

: $A[m]=50 > 48$ – søk til venstre

DFpr(p)
DFpr(left(p))
Skriv(p)
DFpr(right(p))



key(m)=32 < 48 – søk i høyre subtre

key(m)=55 > 48 – søk i venstre subtre

key(m)=50 > 48 – søk i venstre subtre

for alle m, x :

- x i høyre subtre av m : $key(x) \geq key(m)$
- x i venstre subtre av m : $key(x) \leq key(m)$

Binære Søketrær

er et **Binært Tre** T (for lagring av nøkler, eller objekter med nøkler) som tilfredstiller :

BST INVARIANT (“relasjonell”)

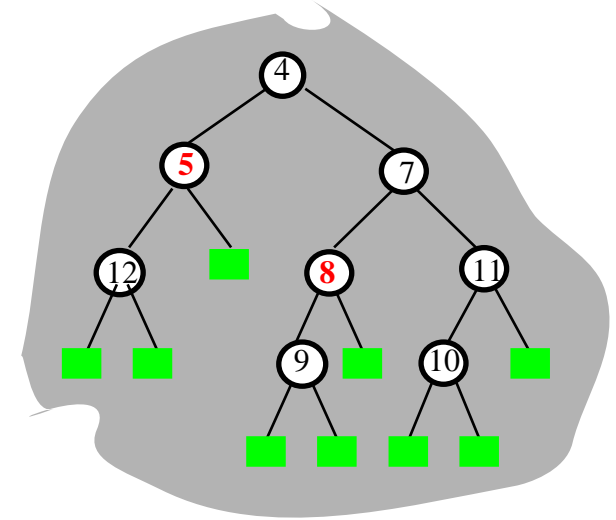
for hver node p :

for enhver node v i p 's **venstre subtre** : $key(v) \leq key(p)$

for enhver node h i p 's **høyre subtre** : $key(h) \geq key(p)$

eller:

DFS (inOrder) enumerering av noder gir en ordnet sekvens



$find(k)$: **findPos**(new Item(k,null),T.root())

findPos (Object ko, Position p)

if (**isExternal**(p)) - k finnes ikke

else if (cp.isEqualTo(ko,key(p))) - funnet

else if (cp.isLessThan(ko,key(p)))

// let videre i venstre subtre

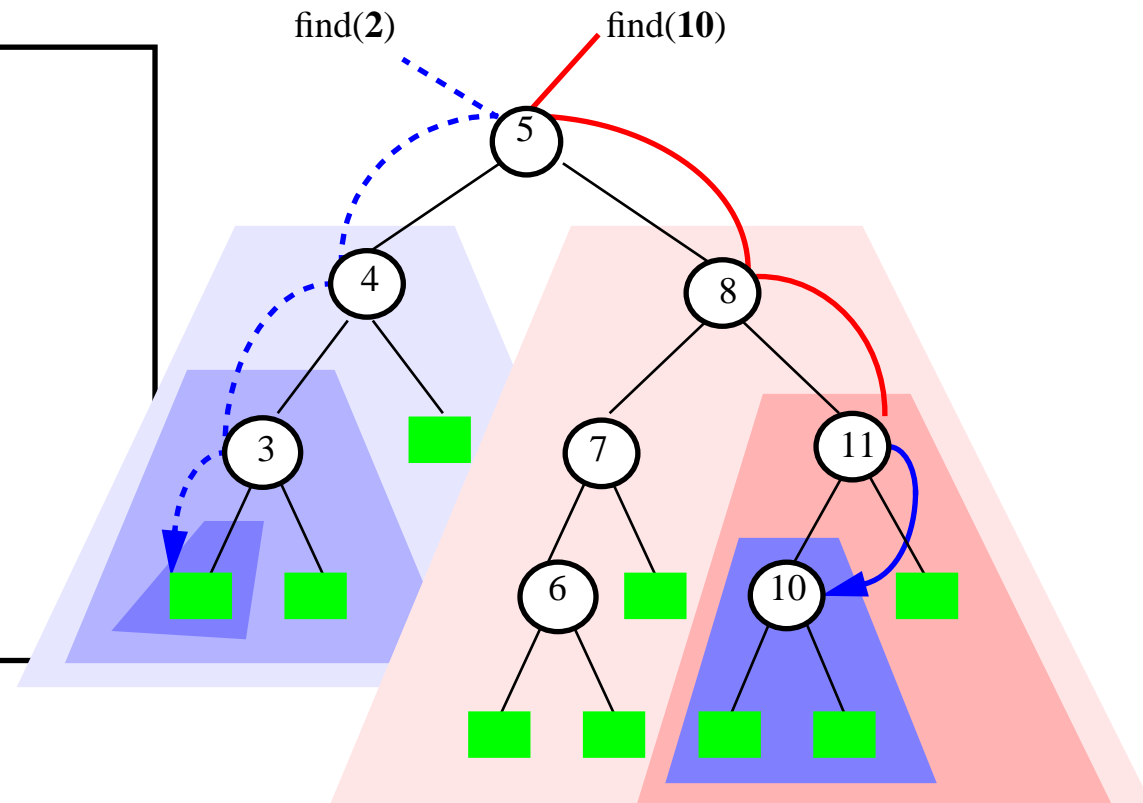
findPos (ko,leftChild(p))

else // cp.isGreaterThan(ko,key(p))

// let videre i høyre subtre

findPos (ko,rightChild(p))

= $O(\text{height}(T))$



Implementasjon av Dictionary – med BST

```
public class BSTDict implements Dictionary {
```

```
protected BinaryTree BST
```



```
protected Comparator cp
```



<, ≤
>, ≥
=, ≠

```
protected boolean DI()
```

```
{ return DI( BST.root(), new Item(-∞, null), new Item(+∞, null) ); }
```

```
protected boolean DI(Position p, Object l, Object h)
```

```
{ if (BST.isExternal(p)) return true;  
  else if ( cp.isLessThan(p.element(), l) || cp.isGreaterThan(p.element(), h) ) return false;  
  else return (DI(BST.leftChild(p), l, key(p)) && DI(BST.rightChild(p), key(p), h)) ; }
```

```
protected Object findPos(Object k) { return findPos(k, BST.root()); }
```

Position **findPos**(Object k, Position p)
{ ... }

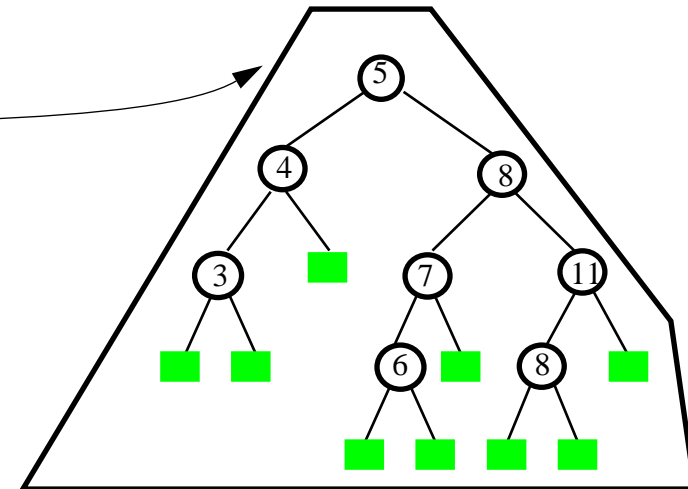
```
public Object find(Object k)
```

```
{ Position p = findPos(k) ;  
  if ( BST.isExternal(p) ) throw new NoSuchKey("...");  
  return ((Item) p.element()).element() ; }
```

```
public ObjectIterator findAll(Object k)
```

```
{ velg en implementasjon av Enumeration E= new Enum();  
  findAll(k, BST.root(), E) ;  
  return E; }
```

```
... insert, remove
```



```
void findAll(Object k, Position p, Iter E) {  
  f= findPos(k,p);  
  if (p != null && BST.isInternal(p))  
    E.add( ((Item)p.element()).element() );  
  findAll(k, BST.leftChild(p), E);  
  findAll(k, BST.rightChild(p), E);  
}
```

insert(Object k, o) i et BST = settInn(BST.root(), k, o)

settInn(Position v, Object k, Object o) {

Position **p** = findPos(k, v)

sikrer
BST INVARIANT

if (**BST.isExternal(p)**)

ny Intern node: expandExternal(**p**)

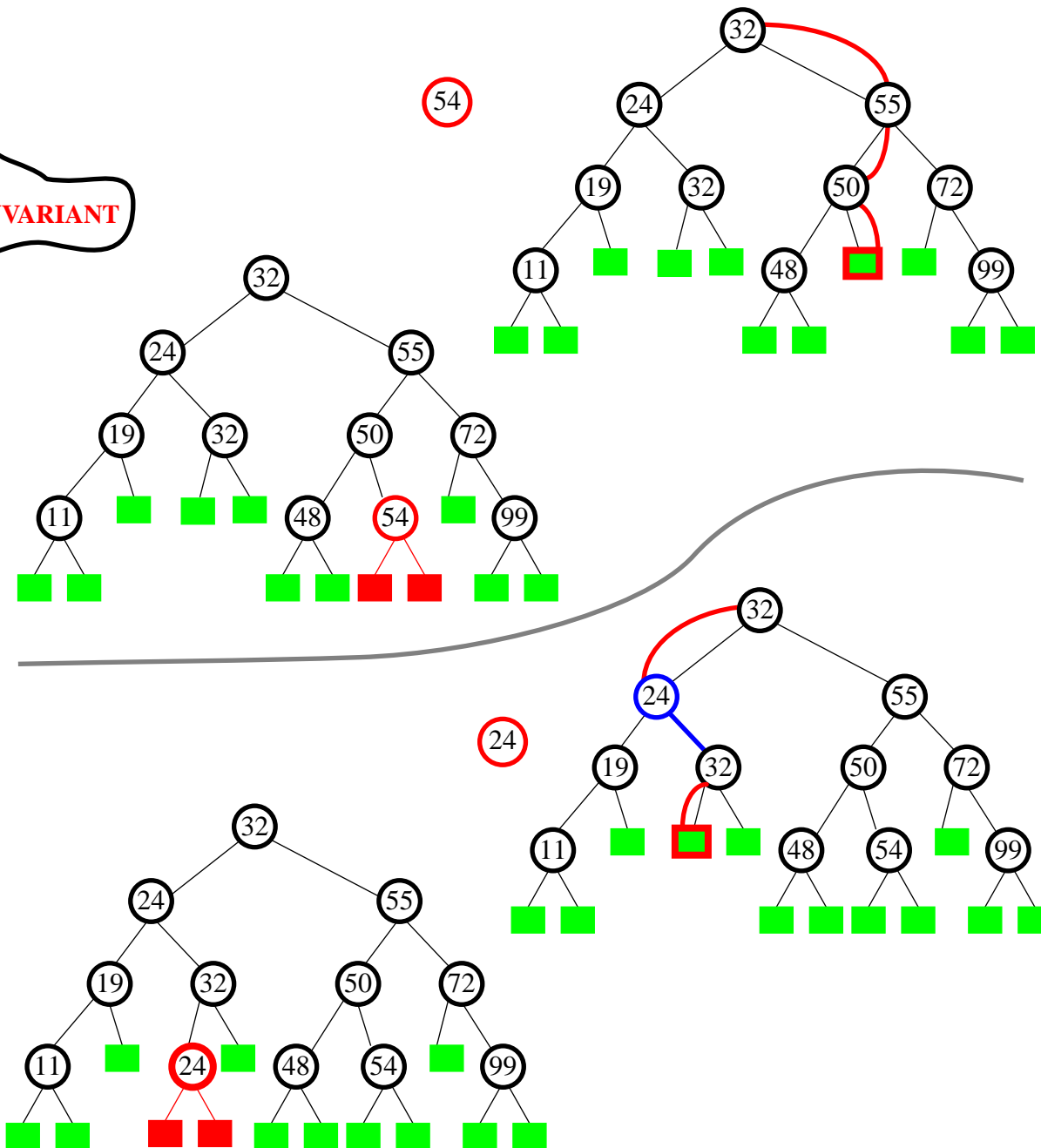
p.setElement(new Item(k,e))

else

settInn(**BST.rightChild(p)**, k, e)

}

= $O(\text{findPos}) = O(h(\text{BST}))$



Object remove(k) fra et BST

Position $p = \text{findPos}(k)$

1. **if** (p er et **blad**) // isExternal(p)

- NoSuchKey

2. **else if** **leftChild**(p) og **rightChild**(p) er **blader**

- fjern p

return ((Item)removeAboveExt(leftChild(p)).element())

3. **else if** **nøyaktig et barn** er et **blad b**

- erstatt p med andre barnet

return ...removeAboveExt(b)...

4. **else** // **ingen av barna** er et **blad**

- hold ((Item)p.element()).element() til return

- finn $n =$ "neste til høyre" for p

$n = \text{rightChild}(p)$

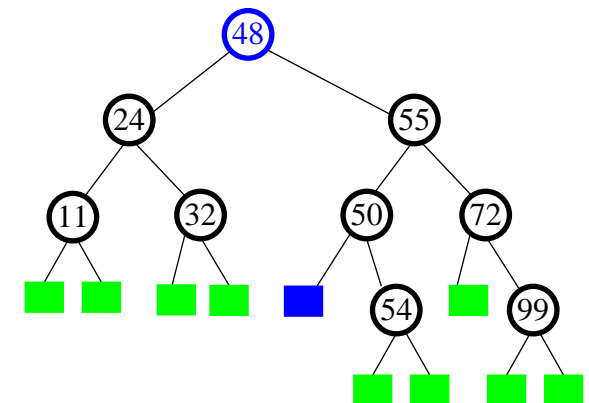
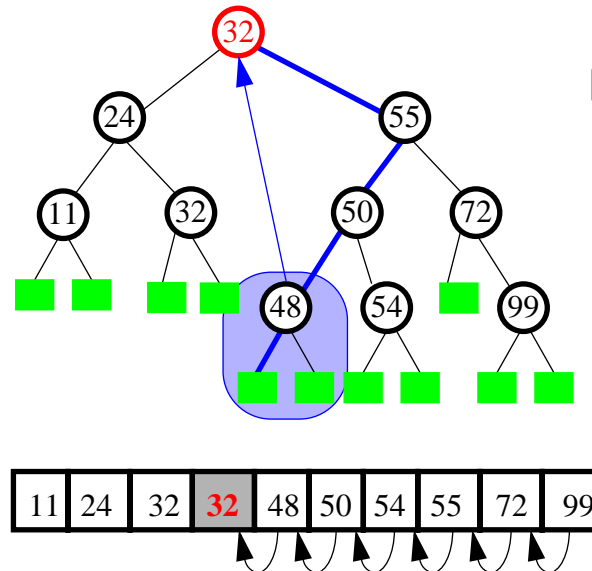
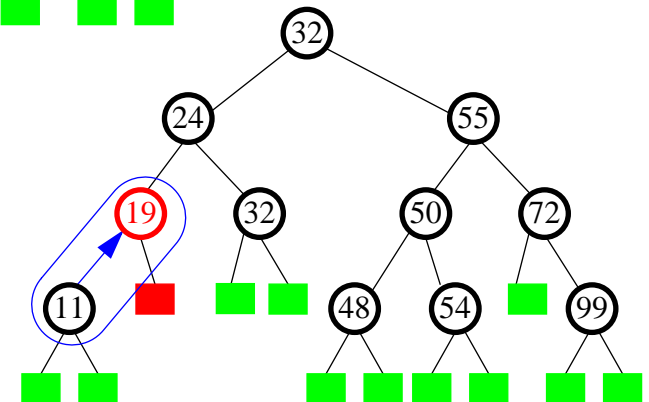
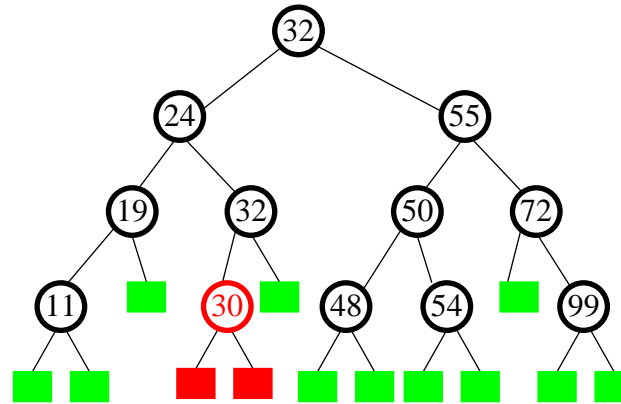
while (! isExternal(leftChild(n)))

$n = \text{leftChild}(n)$

- replace(p , $n.\text{element}()$)

- fjern n (2. eller 3.)

$= O(\text{findPos}) = O(h(\text{BST}))$



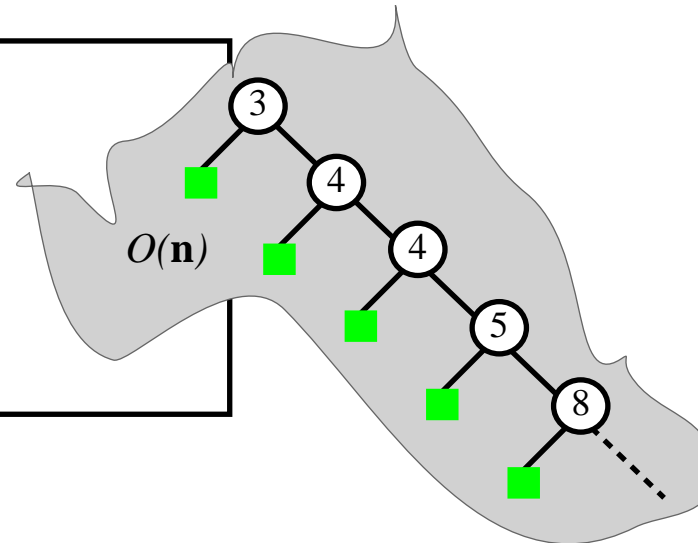
Implementasjon av Dictionary

operasjon	usortert Sequence		sortert Sequence		BST
	Array	DL	Array	DL	
Container					
size	1	1	1	1	1
isEmpty	1	1	1	1	1
elements	n	n	n	n	n
Dictionary					
find	n	n	log n	n	hgh(n)
findAll	n	n	log n (+ s)	n	hgh(n)
insert	1	1	n	n	hgh(n)
remove	n	n	n	n	hgh(n)
closestAfter/Before	n	n	n	n	hgh(n)

Balansering

- hvis BST er balansert får vi $\text{hgh}(n) = O(\log n)$
- dette er det som skjer i gjennomsnittstilfelle (gjennomsnittskompleksitet for BST er $O(\log n)$)

```
settInn(Position v, Object k, Object o) {  
    Position p= findPos(k,v);  
    if (isExternal(p))  
        expandExternal(p)  
        p.setElement(new Item (k,o))  
    else  
        settInn(rightChild(p),k,o) }
```



```
insert(3,A);  
insert(4,B);  
insert(4,B);  
insert(5,C);  
insert(8,D);
```

$\text{hgh}(\text{BST}) = O(n)$

tilsvarende uhell kan skje ved en serie `remove(k)`

AVL Tre

er et Binært Tre T (for lagring av nøkler, eller objekter med nøkler) som tilfredstiller :

1. **BST INVARIANT** (“relasjonell”) – for hver intern node p :
for hver node v i p 's venstre subtre : $\text{key}(v) \leq \text{key}(p)$
for hver node h i p 's høyre subtre : $\text{key}(h) \geq \text{key}(p)$
2. **AVL INVARIANT** (“strukturell”) – hver intern node p er **balansert** :
 $|\text{hgh}(\text{leftChild}(p)) - \text{hgh}(\text{rightChild}(p))| \leq 1$

7.2. Et AVL Tre T som lagrer n nøkler har høyden $h(T) = O(\log n)$

Oppsummering

- *Ordbok (symboltabell)*
avbildning fra nøkler til dataobjekter
- *Ordbok og Ordnet Ordbok ADT*
 - *med nøkkel-data objekter*
 - *med Locator*
- *Implementasjon*
 - Sekvens*
 - *usortert (DL, Array)*
 - *sortert (DL, Array)*
 - Hash Tabeller*
 - *valg av hash-funksjon*
 - *kollisjoner*
 - Binære Søketrær*
 - *innsetting/fjerning*
 - *balansering*