Sekvenser

I. RANKEDSEQUENCE ADT

II. POSISJON & POSSEQUENCE ADT

III. IMPLEMENTASJON

III.1 Array implementasjon

III.2 Liste implementasjon

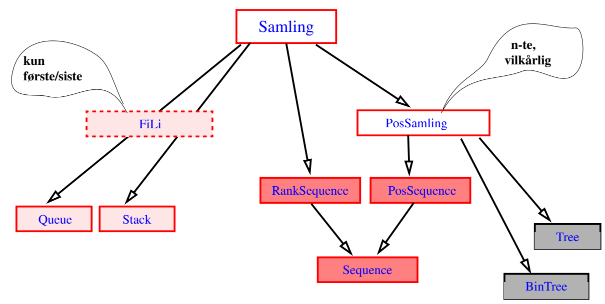
IV. GENERISK SORTERING AV SEKVENSER

V. ITERATOR

Kap. 4 (kursorisk 4.4; + implementasjon av en ADT med en annen ADT)

ADT-hierarki (hittil)

adgang til elementene (innsetting/fjerning/observasjon) avhengig av elementets posisjon i Samlingen



RankSequence ADT

adgang til/fjerning/innsetting av elementer i *vilkårlig* posisjon
(ulik Stack og Queue)

Posisjon – 'rank' – angis her med et naturlig tall 0 <= r < size()

public interface RankSequence extends Samling {

A B C ... V ... Z
0 1 2 ... r ... N

```
/**  setter inn et nytt objekt i posisjon r
 *   @param r  naturlig tall
 *   @param o Objektet som skal innettes
 *   @exception IllegalPosition hvis r < 0 eller r > size()
 */
     void insertElemAtRank(int r, Object o);
```

A B C ... O V ... Z
0 1 2 ... r r+1 ... N+1

```
/**  returnerer objektet i posisjon r
 *   @param r  naturlig tall
 *   @return Objektet  i posisjon r
 *   @exception IllegalPosition hvis r < 0 eller r > size()-1
 */
     Object elemAtRak(int r);

/**  erstatter objektet i posisjon r med et nytt objekt
 *   @param r  naturlig tall
 *   @param u Objektet som skal innettes
 *   @return Objektet  som ble erstattet i posisjon r
 *   @exception IllegalPosition hvis r < 0 eller r > size()-1
 */
     Object replaceElemAtRank(int r, Object u);
```

A B C ... U V ... Z
0 1 2 ... r r+1 ... N+1

```
/**  fjerner og returnerer objekt i posisjon r
 *   @param r  naturlig tall
 *   @return Objektet  som ble fjernet i posisjon r
 *   @exception IllegalPosition hvis r < 0 eller r > size()-1
 */
     Object removeElemAtRank(int r);
}
```

A B C ... V ... Z
0 1 2 ... r ... N

Implementasjon av ADT med ADT

- **DQueue med RankSequence**

A B C ... Z
0 1 2 ... size()-1

```
front()        =  elemAtRank(0)
last()         =  elemAtRank(size()-1)
size()         =  size()
insertFirst(e) =  insertElemAtRank(0,e)
insertLast(e)  =  insertElemAtRank(size(),e)
removeFirst()  =  removeElemAtRank(0)
removeLast()   =  removeElemAtRank(size()-1)
```

```
class DQrs
   implements DQueue {
  private RankSequence s;
  public DQrs(RankSequence r)
    { s= r; }
  ...
}
```

- **EditorLine med RankSequence** (av Character) **+ cur**

DataInvariant: 0 <= cur <= size()

A B C ... V ... Z
0 1 2 ... r ... size()-1
cur=2

```
moveLeft()   =  if (cur > 0) cur--;
moveRight()  =  if (cur < size()) cur++;
moveEnd()    =  cur= size();
moveStart()  =  cur= 0;
insert(c)    =  insertElemAtRank(cur++, new Character(c));
replace(c)   =  replaceElemAtRank(cur, new Character(c));
delete()     =  if (cur < size())  removeElemAtRank(cur);
deleteLeft() =  if (cur > 0)  removeElemAtRank(--cur);
charAt(i)    =  if (0 <= i && i < size())
                  return ((Character)elemAtRank(i)).charValue();
find(c)      =  for (int i=cur; i<size(); i++)
                  if (charAt(i) == c) cur=i; break;
                return cur;
```

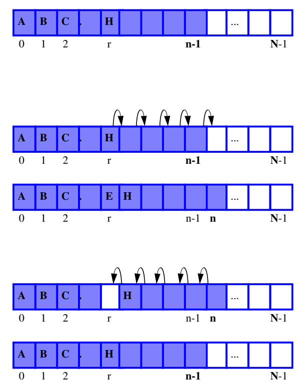
en vilkårlig (ekte) implementasjon av **RankSequence,**
gir umiddelbart en implementasjon av **DQueue, EditorLine**

RankSequence med java.util.Vector

```
public class Vector {
...
int size() { ... }
int capacity() { ... }
void ensureCapacity(int minCapacity) { ... }
void insertElementAt(Object o, int i) { ... }
void setElementAt(Object o, int i) { ... }
void removeElementAt(int i) { ... }
Object lastElement() { ... }
Object firstElement() { ... }
Object elementAt(int i) { ... }
...
}
```

```
import java.util.Vector;
public class RSvector implements RankSequence {
   private Vector v;
   public RSvector() { v= new Vector(); }
   public void insertElemAtRank(int r, Object o) throws IllegalPosition {
      try{ v.insertElementAt(o, r); }
      catch(ArrayIndexOutOfBoundsException e) { throw new IllegalPosition(); }
   public void ElemAtRank(int r) { // sjekk rank r
      return v.elementAt(r);  }
   public Object removeElemAtRank(int r) { // sjekk rank r
      Object o = v.elementAt(r);
      v.removeElementAt(r); return o;  }
   public Object replaceElemAtRank(int r) { // sjekk rank r
      Object o = v.elementAt(r);
      v.setElementAt(r); return o;  }
   public int size() { return v.size();  }
   public boolean empty() { return size()==0; }
}
```

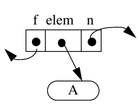

RankedSequence med Array



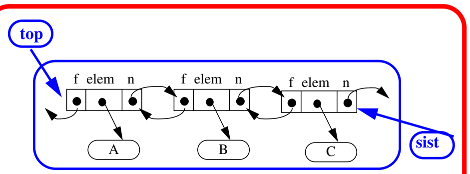

```
private Object[] A; private int n;

int size() { return n; }                          O(1)
bolean isEmpty() { return n==0 }                  O(1)
Object ElemAtRank(r,E) {                          O(1)
   return A[r] }
Object replaceElemAtRank(r,E) {                   O(1)
   e = A[r];   A[r] = E
   return e }

void insertElemAtRank(r,E) {                      O(n)
   for i = n-1, n-2, ... r
      A[i+1] = A[i]
   A[r] = E
   n = n+1
}

Object removeElemAtRank(r) {                      O(n)
   e = A[r]
   for i = r, r+1, ... n-2
      A[i] = A[i+1]
   n = n-1
   return e;
}
```

meget effektive oppslag
men litt mindre effektiv innsetting / fjerning

RankSequence med DL




```
public class DLNode
{     private DLNode f, n;
      private Object elem;
      public DLNode(Object o, DLNode ff, DLNode nn) {
          elem= o; n= nn; f= ff; }
      public Object element() { return elem; }
      public DLNode getNext() { return n; }
      public DLNode getPrev() { return f; }
      public void setElem(Object o) { elem= o; }
      public void setNext(DLNode d) { n= d; }
      public void setPrev(DLNode d) { f= d; }
}
```



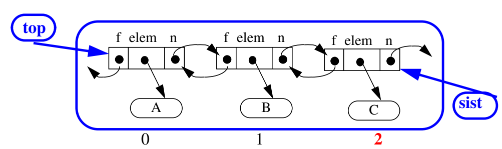
void **insertElemAtRank**(int r, Object o)

Object **removeElemAtRank**(int r)

Object **replaceElemAtRank**(int r, Object o)

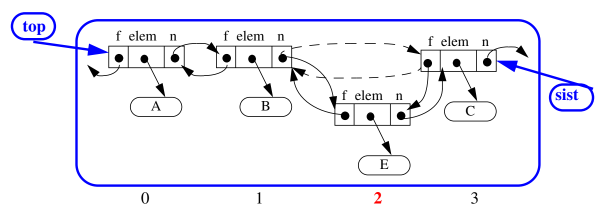
Object **ElemAtRank**(int r)

RankedSequence med DL




```
insertElemAtRank(int r,Object E) {
   DLNode neste= nodeAtRank(r)                  // != null
   DLNode forrige= neste.f                      // != null
   DLNode ny= new DLNode(E,neste,forrige)
   neste.f = ny
   forrige.n= ny }    // ev. oppdater top/sist
```

$\}\ O\,(1)$



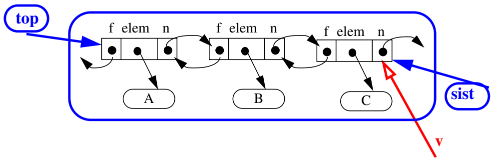

```
removeElemAtRank(int r) {
   DLNode rem= nodeAtRank(r)                    // != null
   if (rem.n != null) rem.n.f= rem.f
   if (rem.f != null) rem.f.n= rem.n
   return rem.getElem() }    // ev. oppd.top/sist
```

$\}\ O\,(1)$

```
private DLNode nodeAtRank(int r) {              // 0 <= r <= size()
     DLNode n= top;
     for (int i=0; i<r; i++) n= n.getNext()
     return n }
```

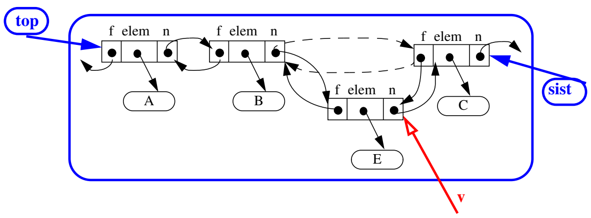
$O\,(n)$

"DL-Sequence"




```
insertBefore(DLNode v,Object E) {          // v != null
    DLNode ny= new DLNode(E, v, v.f)
    v.f.n = ny                             // v.f != null
    v.f= ny }    // ev. oppdater top/sist
```

$O(1)$




```
remove(DLNode v) {                         // != null
    if (v.n != null) v.n.f= v.f
    if (v.f != null) v.f.n= v.n
    return v.getElem() }      // ev. oppd.top/sist
```

$O(1)$

DL-Liste

/** Sekvens der elementene aksesseres gjennom DLNode */
public interface DL-Sequence extends Samling {

		implem.
//	Position aksess	
	DLNode **first**(); // første elem: first().getElem()	**O(1)**
	DLNode **last**();	**O(1)**
	DLNode **before(DLNode v)**;	**O(1)**
	DLNode **after(DLNode v)**;	**O(1)**
//	element håndtering	
/**	@return posisjon av det nye elementet */	
	DLNode **insertFirst(Object e)**;	**O(1)**
/**	@return posisjon av det nye elementet */	
	DLNode **insertLast(Object e)**;	**O(1)**
/**	@return posisjon av det nye elementet */	
	DLNode **insertBefore(DLNode v, Object e)**;	**O(1)**
/**	@return posisjon av det nye elementet */	
	DLNode **insertAfter(DLNode v, Object e)**;	**O(1)**
/**	@return Objektet som ble erstattet */	
	Object **replace(DLNode v, Object e)**;	**O(1)**
/**	@return Objektet som ble fjernet */	
	Object **remove(DLNode v)**;	**O(1)**
/**	bytt om Objekter lagret i v og w*/	
	void **swap(DLNode v, DLNode w)**;	**O(1)**
//	int **size**(); boolean **empty**();	**O(1)**

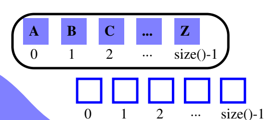
}

Dette er ikke en "riktig" interface fordi den krever en bestemt implementasjon med DL-Liste

Posisjon

(litt spes men ofte nyttig)

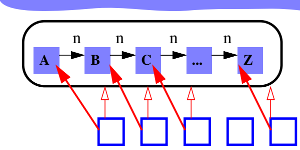
abstraksjon av "et sted"



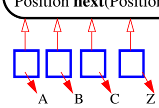
Position = **"et sted" i en struktur som lagrer et Objekt**

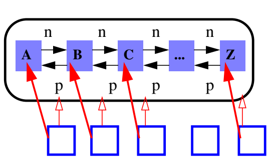
public interface Position {
  Object element();
  Samling container();
  void setElem(Object e); }

"rå struktur" = aksess-metoder til en samling av Position

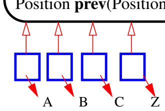


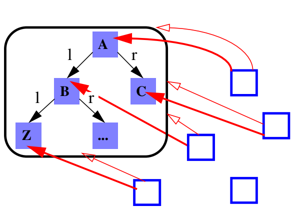
Position **first**()
Position **next**(Position p)



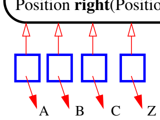


Position **first**(), **last**()
Position **next**(Position p)
Position **prev**(Position p)





Position **first**()
Position **left**(Position p)
Position **right**(Position p)



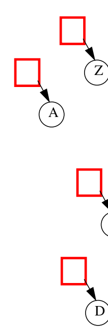
PosSekvens ADT

/** Sekvens der elementene aksesseres gjennom posisjon */
public interface PosSequence extends Samling {

```
int size();
boolean isEmpty();
```

// Position aksess
Position **first**(); // første el.: first().element()
Position **last**();
Position **before**(Position **p**);
Position **after**(Position **p**);

// element håndtering
/** @return posisjon av det nye elementet */
Position **insertFirst(Object e)**;
/** @return posisjon av det nye elementet */
Position **insertLast(Object e)**;
/** @return posisjon av det nye elementet */
Position **insertBefore**(Position **p, Object e)**;
/** @return posisjon av det nye elementet */
Position **insertAfter**(Position **p, Object e)**;
/** @return Objektet som ble erstattet */
Object **replace(**Position **p, Object e)**;
/** @return Objektet som ble fjernet */
Object **remove(**Position **p)**;
/** bytt om */
void **swap(**Position **p,** Position **q)**;
}

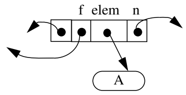


Sequence =
RankedSequence +
PosSequence +
Position **atRank(int r)** +
int **rankOf(Position p)**

Implementasjon med DL krever: **DLNode implements Position**

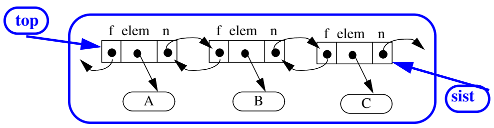
Implementasjon med Array krever: **"Indeks" implements Position**

PosSequence med DL




```
public class DLNode  implements Position
{     private DLNode f, n;
      private Object elem;
      private Samling sam;
      public DLNode(Object o, DLNode ff, DLNode nn, Samling s)
      {   elem= o; n= nn; f= ff; sam= s; }
      public Object element() { return elem; }
      public DLNode getNext() { return n; }
      public DLNode getPrev() { return f; }
      public void setElem(Object o) { elem= o; }
      public void setNext(DLNode d) { n= d; }
      public void setPrev(DLNode d) { f= d; }
      public samling() { return sam; }
}
```

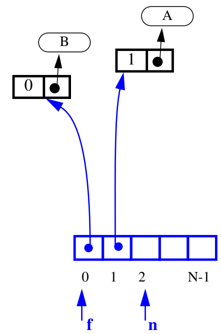
```
public class PosSeqDL implements PosSequence {
```



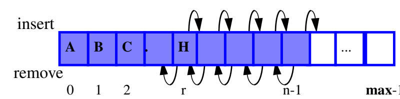

```
      public Position last() { return sist; }
      public Position before(Position p) { return ((DLNode)p).getPrev() ; }
      public Position insertFirst(Object o) {
          top= new DLNode(o, null, top, this) ; return top ; }
      public Position insertBefore(Position p, Object o) {
          DLNode pp = (DLNode)p;
          DLNode ny = new DLNode(o, pp.getPrev(), pp, this);
          return ny; }
. . .  }
```


PosSequence med Array

```
public class ArPos implements Position
{     private int r;
      private Object obj;
      private Samling sam;
      public ArPos(int i, Object o. Samling s)
      {   r= i; obj= o; sam= s; }
      public element() { return obj; }
      public rank() { return r; }
      public setElem(Object o) { obj= o; }
      public container() { return sam; }
      publis setRank(int i) { r= i ; }
}
```




```
public class PosSeqAr implements PosSequence
{     private ArPos[] ar;
      private int f, n, max;
      public Position last() { return ar[n-1]; }
      public Position before(Position p) {
          return ar[ ((ArPos)p).rank()–1 ] ; }
      public Position insertLast(Object o) {
          ar[n] = new ArPos(n,o,this); n++; }
      public Position insertBefore(Position p, Object o) {
          int r = ((ArPos)p).rank();
          for (int k= n; k>r; k--)
              { ar[k]= ar[k–1];  ar[k].setRank(k); }
          ar[r]= new ArPos(r,o,this);
          n++ ;
          return ar[r]; }
. . .
}
```



Sequence implementasjon

operasjon	LL	DL	Array
Samling			
size	1	1	1
isEmpty	1	1	1
PosSequence			
first	1	1	1
last	n / 1	1	1
before	n	1	1
after	1	1	1
insertFirst	1	1	n / 1
insertLast	n / 1	1	1
insertBefore	n	1	n
insertAfter	1	1	n
replace	1	1	1
swap	1	1	1
remove	n	1	n
atRank	n	n	1
rankOf	n	n	1
RankSequence			
elemAtRank	n	n	1
inserElemAtRank	n	n	n
removeElemAtRank	n	n	n
replaceElemAtRank	n	n	1

Sortering

• er en operasjon på strukturer med rekkefølgen på Posisjoner

total ordning: **for alle** Posisjoner p, q : p < q **eller** q < p **eller** q = p

samt before/after operasjoner

Sequence (Pos-, Ranked-, array, liste...)

SelectionSort, MergeSort, ...

• Også elementene lagret i strukturen må stå i en

total ordning: **for alle** Elementer a, b : a < b **eller** b < a **eller** a = b

Designmønster (pattern) tillater å ordne samme Objekter på forskjellige måter

```
public interface Comp {
    boolean lt(Object a, Object b);
    boolean leq(Object a, Object b);
    boolean gt(Object a, Object b);
    boolean geq(Object a, Object b);
    boolean eq(Object a, Object b);
    ... }
```

```
void SS(int[] A) {
  int m, i;
  for (int k=0; k< A.length;k++) {
    m= A[k]; i= k;
    for (int j=k+1;j<A.length;j++) {
      if (A[j] < m) {
          m=A[j]; i= j; }
    swap(A,i,k);
} }
```

```
void SS(Sequence A, Comp C) {
  Object m;
  for (int k=0; k< A.size();k++) {
    m= A.atRank(k).elem(); i= k;
    for (int j=k+1;j<A.size();j++) {
      if (C.lt(A.atRank(j).elem(),m) {
          m=A.atRank(j).elem(); i= j; }
    A.swap(A.atRank(i), A.atRank(k));
} }
```

Generisk SeleksjonSort av Sequence

```
*   for (k=0,1,2...< n) {    // n= S.size()
*       int m = k;
*       for (j= k+1...< n) {
*           if (S[j] < S[m]) m = j;
*       S.swap(m,k)
*       }

*   void SS1(Sequence S, Comp C)   int n = S.size()
*       int k, j, min;
*       for (k=0,1,2...< n)  {
*           min = k ; em = S.atRank(min).elem();
*           for (j= k+1...< n)
*              if (C.lt( S.atRank(j).elem(), em ))
*                    { min = j; em = S.atRank(min).elem(); }
*           S.swap(S.atRank(min), S.atRank(k)) }
*
```

kan brukes kun når **atRank(r)** er **O(1)** ; er den **O(n)** får vi SS1 = **O(n^3)** !!!

```
*   void SS2(Sequence S, Comp C)
*       Position k, j, min;
*       k= S.first();
*       while (k != S.last())   {
*           min= k; j= k;
*           while (j != S.last()) {
*               j= S.after(j);
*               if (C.lt (j.elem(), m.elem()) ) min = j; }
*           S.swap(min, k)
*           k= S.after(k);   }
```

first(), last() og **after(p)** kan alltid implementeres med **O(1)**

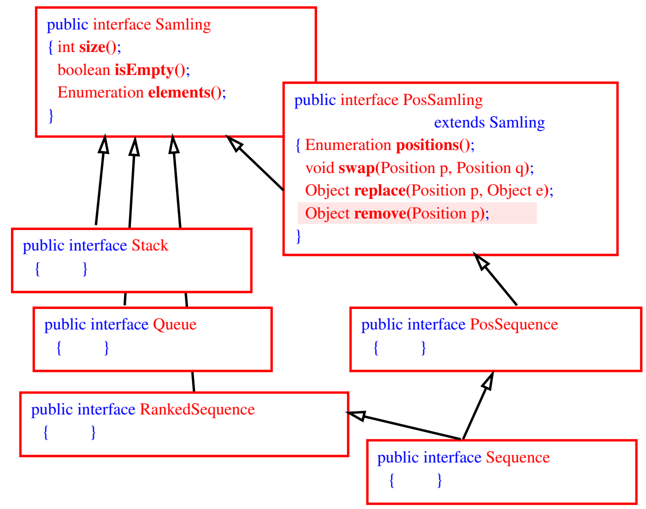
Iterator

fra en "uordnet" samling til en "sekvens-aktig" samling

med "current element" og "next element"

```
public interface Enumeration {
    boolean hasMoreElements();
    Object nextElement();
}
```

brukes for å traversere alle elementene én gang !!!



BubbleSort

```
/*  BS - sorterer input Sequence med Comp:
 *      @param S - sekevens som skal sorteres
 *      @param C - sammenligningsoperator
 *      @return - sortert input sekvens
 *
```

4↔1	3	**4** ↔	1	0	5	2	i=0
4↔0	3	1	**4** ↔	0	5	2	i=0
5↔2	3	1	0	4	**5** ↔	2	i=0
3↔1 3↔0 4↔2	**3**	1	0	**4**	2	5	i=1
1↔0 3↔2	**1**	0	**3**	2	4	5	i=2
	0	1	2	3	4	5	i=3

```
 *
 *  for (i=0,1,2...< n) {    // n= S.size()
 *      for (j=0,1,2...< n-i-1) {
 *          if ( C.lt( S[j+1] , S[j] ))  S.swap(j,j+1)
 *      }
 */
```

Invariant:

etter *i'te* passering er
***(n-i)'te* elementet – og alle fom. dette tom. *n'te* – riktig plassert**

$$BS(n) \;=\; O(\, n + (n\text{-}1) + (n\text{-}2) + \ldots\ 1\,) \qquad = O\,(n^2)$$
$$= \Theta\,(n^2)$$