Høgskolen i Sørøst-Norge

EKSAMEN

5610

ALGORITMER OG DATASTRUKTURER

18.05.2016

Tid:	5 timer (9-14)
Målform:	Nynorsk og bokmål
Sidetal:	11 (forside + 3 + 3 + 4)
Hjelpemiddel:	Alle trykte og skrevne
Merknader:	Ingen
Vedlegg:	Dokumentasjon av LinkedList og ListIterator-klassene

Sensuren finner du på StudentWeb.

Råd og retningslinjer. Les oppgaveteksten godt før du går i gang med å løse oppgaven. Deloppgavene er uavhengige av hverandre i den forstand at om du ikke får til en oppgave, kan du likevel gjøre neste, som om den første var løst. Fordél tiden godt på alle oppgavene. Om du mener en oppgave er upresis, så skriv din egen presisering. Lag de hjelpemetoder du måtte ønske.

Innledning

Dersom du trenger en datastruktur som kan holde på vilkårlig mange objekter, og returnere eller slette *det minste* på en effektiv måte, har vi sett at Binary Heap er et godt valg. Om du trenger tilgang til *det største* istedenfor det minste, er det lett å tilpasse Binary-Heap-algoritmene. Men hva om du trenger tilgang til både det minste og det største? Det er utgangspunktet for disse oppgavene. En struktur som støtter dette kalles gjerne en DEPQ – Double-Ended Priority Queue.

Oppgave 1 (40%)

Følgende grensesnitt er laget:

```
public interface DEPQ<AnyType extends Comparable<? super AnyType>>{
 void add(AnyType x);      // legger inn objektet x
 AnyType findMin();        // returnerer minste objekt
 AnyType removeMin();      // sletter og returnerer minste objekt
 AnyType findMax();        // returnerer største objekt
 AnyType removeMax();      // sletter og returnerer største objekt
 int size();               // returnerer antall objekter
 boolean isEmpty();        // true dersom tom
 void makeEmpty();         // sletter alt
}
```

En enkel og grei løsning er å lage en datastruktur basert på tabell. Den skal holde objektene sortert, den skal fungere som en ringbuffer, og «vokse» automatisk ved behov. Du skal i alle deloppgaver skrive all kode som behøves, ikke henvise til læreboka. Trenger du hjelpemetoder så skriv også disse.

1a (ca 15%)

Skriv klasseheading for class ArrayMinMax, den skal implementere DEPQ. Skriv deklarasjon av de instansvariable du behøver, to konstruktørmetoder (uten og med kapasitetsparameter), size, isEmpty og makeEmpty. Den siste skal slette fysisk, ikke bare logisk.

1b (ca 10%)

Skriv metodene findMin, removeMin, findMax og removeMax.

1c (ca 10%)

Skriv metoden add. Dersom tabellen er full må den «utvides», dvs. lage ny, større.

1d (ca 5%)
Hvilken orden har metodene size, isEmpty, makeEmpty, add, findMin, removeMin, findMax og removeMax? Gi kort begrunnelse.

Oppgave 2 (25%)

Istedenfor å bruke tabell, kan man tenke seg å bruke en sortert pekerkjede.

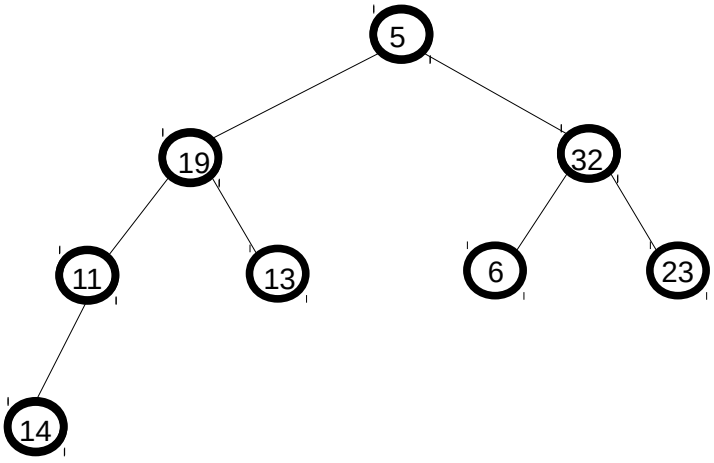
2a (ca 15%)
Lag en klasse LinkedMinMax som implementerer DEPQ og som har en LinkedList som instansvariabel. Se vedlegg. Lag parameterløs konstruktørmetode og sju av de åtte metodene i DEPQ: size, isEmpty, makeEmpty, findMin, removeMin, findMax og removeMax.

2b (ca 10%)
Lag add-metoden.

Oppgave 3 (35%)

En mer avansert datastruktur vi kan bruke til å implementere DEPQ, er en såkalt Min-max Heap. Det er en variant av Binary Heap med følgende egenskaper:

- Komplett binærtre: Alle nivå er fulle unntatt nederste som er fylt fra venstre
- Annethvert nivå er et min-nivå og et max-nivå. Rotnoden er på min-nivå, dens barn er på max-nivå, dens barnebarn er på min-nivå, osv.
- Et element på min-nivå er mindre enn eller lik alle sine etterkommere
- Et element på max-nivå er større enn eller lik alle sine etterkommere



Figur 1: Eksempel på Min-max Heap

Figur 1 viser et eksempel på en Min-max Heap. Vi ser at rotnoden er på min-nivå, den er mindre enn alle sine etterkommere. Barna (19 og 32) utgjør et max-nivå, de er større enn alle sine respektive etterkommere. Neste nivå (11, 13, 6 og 23) er et min-nivå, osv.

3a (ca 10%)

Gitt verdiene 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Tegn en Min-max Heap som inneholder disse verdiene, på kladd (det er mange løsninger!). Skriv så i din besvarelse elementene i den Min-max Heapen i preorden, inorden, postorden og nivåorden rekkefølge.

3b (ca 8%)

Som vi har sett i forbindelse med Binary Heap er det mest effektivt å implementere strukturen ved hjelp av tabell. Lag klasseheading for class HeapMinMax som også skal implementere DEPQ, deklarer nødvendige instansvariable for tabellrepresentasjon, lag to konstruktørmetoder (uten og med kapasitetsparameter), og metodene size, isEmpty og makeEmpty. I denne klassen velger vi å utføre sletting bare logisk, ikke fysisk.

3c (ca 7%)

Implementer metodene findMin og findMax i class HeapMinMax. Du må selv tenke ut hvor max-elementet befinner seg.

3d (ca 10%)

Metodene add, removeMin (og removeMax) ligner på de tilsvarende metodene i class BinaryHeap, men de er noe mer komplisert, så de skal *ikke* lages her. Isteden skal du lage en metode

```
public boolean testMinMax()
```

i class HeapMinMax som tester om dette objektet faktisk tilfredsstiller minmax-egenskapen, dvs. at noder på min-nivå er mindre enn eller lik alle sine etterkommere, noder på max-nivå er større enn eller lik alle sine etterkommere, annethvert nivå i binærtreet (på tabellform) er min- og maxnivå, og rotnoden er på min-nivå. Du skal ikke teste om komplett-egenskapen er tilfredsstilt. Hint: lag to rekursive hjelpemetoder, testMin og testMax, som henholdsvis tester om en node er en min-node og om en node er en max-node. De trenger parametre som angir grensene for hvilket intervall noden må ligge i. Det vil her være hensiktsmessig å bruke indirekte rekursjon.

Lykke til!

Råd og retningsliner. Les oppgåveteksten godt før du går i gang med å løyse oppgåva. Deloppgåvene er uavhengige av kvarandre i den meining at om du ikkje får til ei oppgåve, kan du likevel gjere neste, som om den fyrste var løyst. Fordél tida godt på alle oppgåvene. Om du meiner ei oppgåve er upresis, så skriv din eigen presisering. Lag dei hjelpemetodar du måtte ynskje.

Innleiing

Dersom du treng ein datastruktur som kan halde på vilkårleg mange objekt, og returnere eller slette *det minste* på ein effektiv måte, har vi sett at Binary Heap er eit godt val. Om du treng tilgang til *det største* istaden for det minste, er det lett å tilpasse Binary-Heap-algoritmene. Men kva om du treng tilgang til både det minste og det største? Det er utgangspunktet for desse oppgåvene. Ein struktur som støtter dette kallast gjerne ein DEPQ – Double-Ended Priority Queue.

Oppgåve 1 (40%)

Fylgjande grensesnitt er laga:

```
public interface DEPQ<AnyType extends Comparable<? super AnyType>>{
 void add(AnyType x);      // legg inn objektet x
 AnyType findMin();        // returnerer minste objekt
 AnyType removeMin();      // slettar og returnerer minste objekt
 AnyType findMax();        // returnerer største objekt
 AnyType removeMax();      // slettar og returnerer største objekt
 int size();               // returnerer antal objekt
 boolean isEmpty();        // true dersom tom
 void makeEmpty();         // slettar alt
}
```

Ei enkel og grei løysing er å lage ein datastruktur basert på tabell. Den skal halde objekta sortert, den skal fungere som ein ringbuffer, og «vekse» automatisk ved behov. Du skal i alle deloppgåver skrive all kode som du treng, ikkje vise til læreboka. Treng du hjelpemetodar så skriv også desse.

1a (ca 15%)

Skriv klasseheading for class ArrayMinMax, den skal implementere DEPQ. Skriv deklarasjon av dei instansvariable du treng, to konstruktørmetodar (utan og med kapasitetsparameter), size, isEmpty og makeEmpty. Den siste skal slette fysisk, ikkje berre logisk.

1b (ca 10%)

Skriv metodane findMin, removeMin, findMax og removeMax.

1c (ca 10%)

Skriv metoden add. Dersom tabellen er full må den «utvidast», dvs. lage ny, større.

1d (ca 5%)
Kva for orden har metodane size, isEmpty, makeEmpty, add, findMin, removeMin, findMax og removeMax? Gje kort grunngjeving.

Oppgåve 2 (25%)

Istaden for å nytte tabell, kan ein tenkje seg å nytte ein sortert peikarkjede.

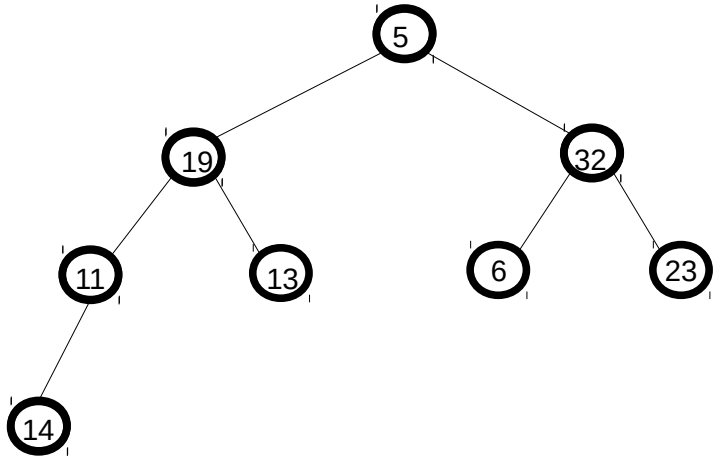
2a (ca 15%)
Lag ei klasse LinkedMinMax som implementerer DEPQ og som har ei LinkedList som instansvariabel. Sjå vedlegg. Lag parameterlaus konstruktørmetode og sju av dei åtte metodane i DEPQ: size, isEmpty, makeEmpty, findMin, removeMin, findMax og removeMax.

2b (ca 10%)
Lag add-metoden.

Oppgave 3 (35%)

Ein meir avansert datastruktur vi kan nytte til å implementere DEPQ, er ein såkalla Min-max Heap. Det er ein variant av Binary Heap med fylgjande eigenskapar:

- Komplett binærtre: Alle nivå er fulle unntatt nederste som er fylt frå venstre
- Annakvart nivå er eit min-nivå og eit max-nivå. Rotnoden er på min-nivå, barna til rotnoden er på max-nivå, barnebarna er på min-nivå, osv.
- Eit element på min-nivå er mindre enn eller lik alle sine etterkommarar
- Eit element på max-nivå er større enn eller lik alle sine etterkommarar



Figur 2: Døme på Min-max Heap

Figur 2 viser eit døme på ein Min-max heap. Vi ser at rotnoden er på min-nivå, den er mindre

enn alle sine etterkommarar. Barna (19 og 32) utgjer eit max-nivå, dei er større enn alle sine respektive etterkommarar. Neste nivå (11, 13, 6 og 23) er eit min-nivå, osv.

3a (ca 10%)

Gitt verdiane 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Teikn ein Min-max Heap som inneheld desse verdiane, på kladd (det er mange løysingar!). Skriv så i ditt svar elementa i den Min-max Heapen i preorden, inorden, postorden og nivåorden rekkefølgje.

3b (ca 8%)

Som vi har sett i samband med Binary Heap er det mest effektivt å implementere strukturen ved hjelp av tabell. Lag klasseheading for class HeapMinMax som også skal implementere DEPQ, deklarer naudsynte instansvariable for tabellrepresentasjon, lag to konstruktørmetodar (utan og med kapasitetsparameter), og metodane size, isEmpty og makeEmpty. I denne klassa velgjer vi å utføre sletting berre logisk, ikkje fysisk.

3c (ca 7%)

Implementer metodane findMin og findMax i class HeapMinMax. Du må sjøl tenkje ut kvar du finn max-elementet.

3d (ca 10%)

Metodane add, removeMin (og removeMax) liknar på dei tilsvarande metodane i class BinaryHeap, men dei er noko meir komplisert, så dei skal *ikkje* lagast her. Istaden skal du lage ein metode

```
public boolean testMinMax()
```

i class HeapMinMax som testar om dette objektet faktisk stettar minmax-eigenskapen, dvs. at nodar på min-nivå er mindre enn eller lik alle etterkommarane sine, nodar på max-nivå er større enn eller lik alle etterkommarane sine, annakvart nivå i binærtreet (på tabellform) er min- og maxnivå, og rotnoden er på min-nivå. Du skal ikkje teste om komplett-eigenskapen er tilfredsstilt. Hint: lag to rekursive hjelpemetodar, testMin og testMax, som høvesvis testar om ein node er ein min-node og om ein node er ein max-node. Dei treng parametre som gjev grensene for intervallet noden må ligge i. Det vil her vera turvande å nytte indirekte rekursjon.

Lykke til!

LinkedList Methods

Modifier and Type	Method and Description
boolean	**add**(E e) Appends the specified element to the end of this list.
void	**add**(int index, E element) Inserts the specified element at the specified position in this list.
boolean	**addAll**(Collection<? extends E> c) Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator.
boolean	**addAll**(int index, Collection<? extends E> c) Inserts all of the elements in the specified collection into this list, starting at the specified position.
void	**addFirst**(E e) Inserts the specified element at the beginning of this list.
void	**addLast**(E e) Appends the specified element to the end of this list.
void	**clear**() Removes all of the elements from this list.
Object	**clone**() Returns a shallow copy of this LinkedList.
boolean	**contains**(Object o) Returns true if this list contains the specified element.
Iterator<E>	**descendingIterator**() Returns an iterator over the elements in this deque in reverse sequential order.
E	**element**() Retrieves, but does not remove, the head (first element) of this list.
E	**get**(int index) Returns the element at the specified position in this list.
E	**getFirst**() Returns the first element in this list.
E	**getLast**() Returns the last element in this list.
int	**indexOf**(Object o) Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element.
int	**lastIndexOf**(Object o)

	Returns the index of the last occurrence of the specified element in this list, or -1 if this list does not contain the element.
ListIterator<E>	**listIterator**(int index) Returns a list-iterator of the elements in this list (in proper sequence), starting at the specified position in the list.
boolean	**offer**(E e) Adds the specified element as the tail (last element) of this list.
boolean	**offerFirst**(E e) Inserts the specified element at the front of this list.
boolean	**offerLast**(E e) Inserts the specified element at the end of this list.
E	**peek**() Retrieves, but does not remove, the head (first element) of this list.
E	**peekFirst**() Retrieves, but does not remove, the first element of this list, or returns null if this list is empty.
E	**peekLast**() Retrieves, but does not remove, the last element of this list, or returns null if this list is empty.
E	**poll**() Retrieves and removes the head (first element) of this list.
E	**pollFirst**() Retrieves and removes the first element of this list, or returns null if this list is empty.
E	**pollLast**() Retrieves and removes the last element of this list, or returns null if this list is empty.
E	**pop**() Pops an element from the stack represented by this list.
void	**push**(E e) Pushes an element onto the stack represented by this list.
E	**remove**() Retrieves and removes the head (first element) of this list.
E	**remove**(int index) Removes the element at the specified position in this list.
boolean	**remove**(Object o) Removes the first occurrence of the specified element from this list, if it is present.
E	**removeFirst**() Removes and returns the first element from this list.

boolean	**removeFirstOccurrence**(Object o) Removes the first occurrence of the specified element in this list (when traversing the list from head to tail).
E	**removeLast**() Removes and returns the last element from this list.
boolean	**removeLastOccurrence**(Object o) Removes the last occurrence of the specified element in this list (when traversing the list from head to tail).
E	**set**(int index, E element) Replaces the element at the specified position in this list with the specified element.
int	**size**() Returns the number of elements in this list.
Object[]	**toArray**() Returns an array containing all of the elements in this list in proper sequence (from first to last element).
<T> T[]	**toArray**(T[] a) Returns an array containing all of the elements in this list in proper sequence (from first to last element); the runtime type of the returned array is that of the specified array.

- **Methods inherited from class java.util.AbstractSequentialList**

  iterator

- **Methods inherited from class java.util.AbstractList**

  equals, hashCode, listIterator, removeRange, subList

- **Methods inherited from class java.util.AbstractCollection**

  containsAll, isEmpty, removeAll, retainAll, toString

- **Methods inherited from class java.lang.Object**

  finalize, getClass, notify, notifyAll, wait, wait, wait

- **Methods inherited from interface java.util.List**

  containsAll, equals, hashCode, isEmpty, iterator, listIterator, removeAll, retainAll, subList

- **Methods inherited from interface java.util.Deque**

  iterator

ListIterator Methods

Modifier and Type	Method and Description
`void`	**`add`**`(E e)` Inserts the specified element into the list (optional operation).
`boolean`	**`hasNext`**`()` Returns `true` if this list iterator has more elements when traversing the list in the forward direction.
`boolean`	**`hasPrevious`**`()` Returns `true` if this list iterator has more elements when traversing the list in the reverse direction.
`E`	**`next`**`()` Returns the next element in the list and advances the cursor position.
`int`	**`nextIndex`**`()` Returns the index of the element that would be returned by a subsequent call to `next()`.
`E`	**`previous`**`()` Returns the previous element in the list and moves the cursor position backwards.
`int`	**`previousIndex`**`()` Returns the index of the element that would be returned by a subsequent call to `previous()`.
`void`	**`remove`**`()` Removes from the list the last element that was returned by `next()` or `previous()` (optional operation).
`void`	**`set`**`(E e)` Replaces the last element returned by `next()` or `previous()` with the specified element (optional operation).