



## Programmering för Språkteknologer II

Markus Saers  
markus.saers@lingfil.uu.se  
Rum 9-2040  
stp.lingfil.uu.se/~markuss/ht09/pst2



## Innehåll

- Dubbellänkade listor
- Associativa datastrukturer
  - Hashtabeller
  - Sökträd
- Implementationsdetaljer för att använda associativa datastrukturer

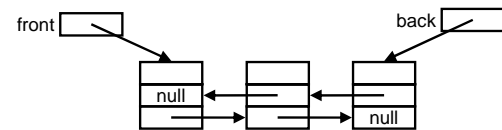


## Dubbellänkade listor

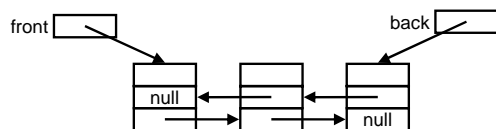
- Insättning
- Borttagning



## Dubbellänkad lista



## Dubbellänkad lista

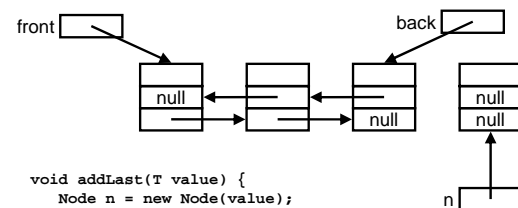


```
void addLast(T value) {
}

```



## Dubbellänkad lista



```
void addLast(T value) {
    Node n = new Node(value);
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    back.next = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    back.next = n;
    back = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    back.next = n;
    back = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkade listor

- Insättning
- Borttagning

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void removeLast() {
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void removeLast() {
    back.prev.next = null;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void removeLast() {
    back.prev.next = null;
    back = back.prev;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void removeLast() {
    back.prev.next = null;
    back = back.prev;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void removeLast() {
    back.prev.next = null;
    back = back.prev;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void removeLast() {
    if (back.prev != null) {
        back.prev.next = null;
        back = back.prev;
    } else {
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

```

void removeLast() {
    if (back.prev != null) {
        back.prev.next = null;
        back = back.prev;
    } else {
        back = null;
        front = null;
    }
}

```



UPPSALA UNIVERSITET

### Dubbellänkad lista

front  back

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    back.next = n;
    back = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

front  back

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    n.prev = back;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

front  back

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

front  back

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

front  back

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

### Dubbellänkad lista

front  back

```

void addLast(T value) {
    Node n = new Node(value);
    Node n = new Node(value);
    n.prev = back;
    n.prev = back;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

## Dubbellänkad lista

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

## Dubbellänkad lista

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    n.next = null;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

## Dubbellänkad lista

```

void addLast(T value) {
    Node n = new Node(value);
    n.prev = back;
    if (back != null) back.next = n;
    else front = n;
    back = n;
}

```

UPPSALA UNIVERSITET

## Frågor?

UPPSALA UNIVERSITET

## Associativa datastrukturer

- Låter oss associera godtyckliga nycklar med godtyckliga värden
  - Vektor: associerar heltal med godtyckliga värden
- Behöver inte vara "tät"
  - Vektorer: måste ha plats för alla nycklar från 0 till den högsta
- Nycklar måste vara unika
  - En nyckel får inte associeras med flera värden
  - Utgör en "mängd"
    - Vektorer: mängden giltiga index

UPPSALA UNIVERSITET

## Associativa datastrukturer

- Två vanliga implementationer
  - Hashtabeller
    - (jätte) snabba
      - $O(1)$
    - Osorterade
    - Nästan täta
  - Sökträd
    - Snabba
      - $O(\log n)$
    - Sorterade
    - Helt täta

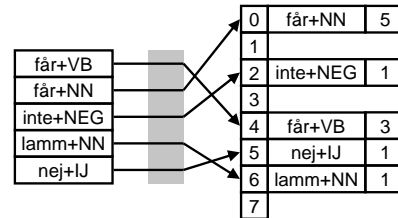


## Hashtabeller

- Vektor med nyckel-värde par
- Hashfunktion
  - Omvandlar nycklar till giltiga index



## Hashtabeller

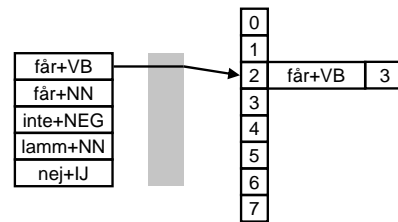


## Hashtabeller

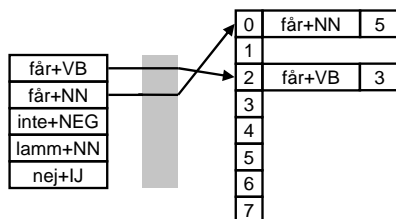
- Sökning
  - Tillämpa hashfunktionen
  - Slå upp det erhållna indexet
- Krockhantering
  - Vad händer om två värden får samma index?



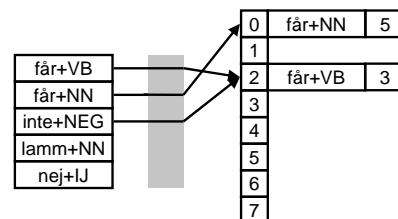
## Hashtabeller: krockhantering

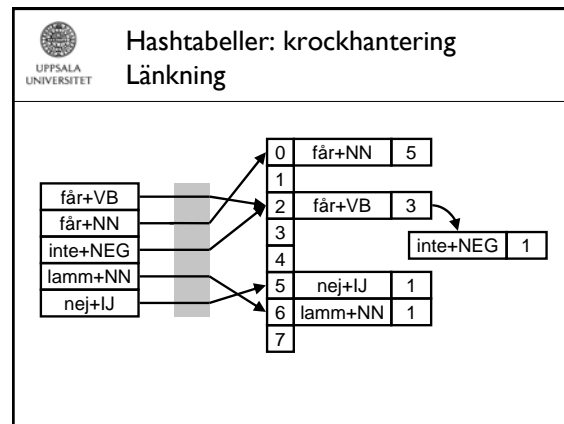
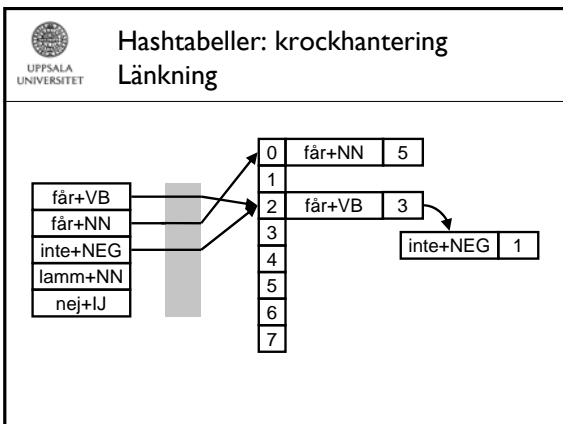


## Hashtabeller: krockhantering



## Hashtabeller: krockhantering

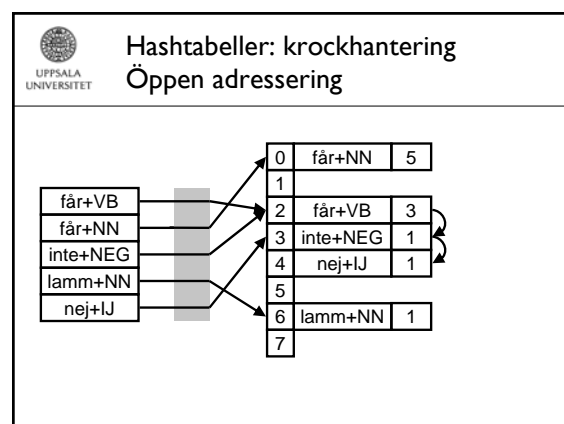
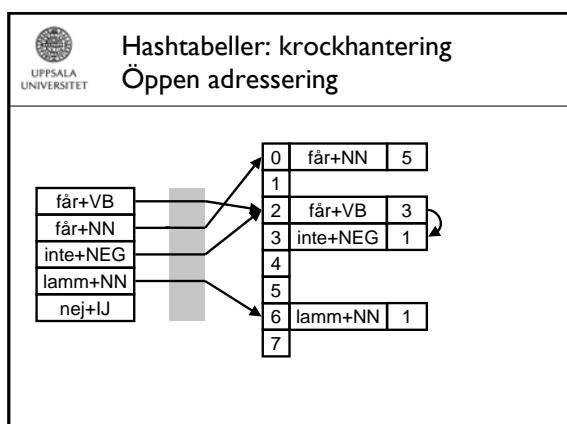
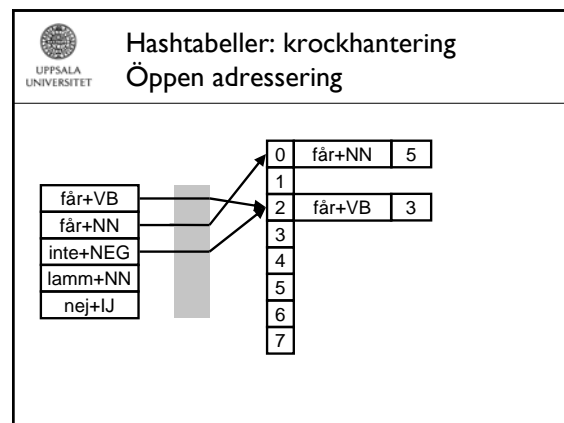




UPPSALA UNIVERSITET

### Krockhantering: Länkning

- Varje plats i vektorn är en länkad lista med nyckel-värde-par
- Sökning
  - Applicera hashfunktionen på nyckeln
  - Löp igenom listan tills
    - Nyckeln hittas (medlem i nyckelmängden)
    - Slutet på listan nås (ej medlem i nyckelmängden)





## Krockhantering: Öppen adressering

- Sökning
  - Applicera hashfunktionen på nyckeln
  - Om platsen är upptagen av en annan nyckel
    - Gå till en annan plats i vektorn
    - Repetera tills nyckeln eller en tom plats hittas
- Olika strategier
  - Gå ett steg i taget
    - Linjär sökning
  - Öka steglängden med antalet steg som tagits
    - Kvadratisk sökning
  - Låt en hashfunktion avgöra hur många steg som ska tas
    - Dubbelhashning



## Täthet

- Ju fler nycklar som finns i en hashtabell desto större är risken för krockar
- Ju fler krockar desto längre tid tar insättning/sökning
- Trade-off mellan tid och utrymme
- Täthet: antal nycklar/plats
  - 0,75 bra riktvärde



## I Java

- Klasserna **HashMap** och **HashSet**
  - En mängd kan modelleras genom att man ignorerar värdena och bara bryr sig om nyckelmängden i den associativa datastrukturen
  - Använder länkning som krockhantering
- Kräver en implementation av metoderna **int hashCode()** och **boolean equals(Object other)**
  - Avgör vilket värde som ska in i hashfunktionen
  - Avgör ifall två nycklar är lika
  - Finns definierad i **Object**



## Konstruktörer i HashMap och HashSet

- **HashXXX()**
  - Sätter `initialCapacity` till 16
  - Sätter `loadFactor` till 0,75
- **HashXXX(int initialCapacity)**
  - Sätter `loadFactor` till 0,75
- **HashXXX(int initialCapacity, float loadFactor)**
- När antalet element överstiger kapaciteten gånger täthetsfaktorn ökas utrymmet
  - Tar tid, inte bra att göra ofta
- Täthetsfaktorn kan användas för att balansera effektivitet mot utrymme



## Krockhantering

- Javas implementation av **HashMap** och **HashSet** hanterar krockar med länkning
- Metoden **int hashCode()**
  - Avgör vilken plats en nyckel ska in på
- Metoden **boolean equals(Object o)**
  - Avgör om två nycklar är lika i länknigen
- Båda måste implementeras för att objekt av en klass ska kunna hashas



## Idén med hashCode()

- Whenever it is invoked on the same object more than once during an execution of a Java application, the **hashCode** method must consistently return the same integer, provided no information used in **equals** comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.



### Idén med hashCode ( )

- If two objects are equal according to the `equals(Object)` method, then calling the `hashCode` method on each of the two objects must produce the same integer result.



### Idén med hashCode ( )

- It is *not* required that if two objects are unequal according to the `equals(java.lang.Object)` method, then calling the `hashCode` method on each of the two objects must produce distinct integer results. However, the programmer should be aware that producing distinct integer results for unequal objects may improve the performance of hashtables.



### Idén med hashCode ( )

- Två objekt som är lika enligt `equals` ska ge samma `hashCode`
- Två objekt som inte är lika enligt `equals` bör ge olika `hashCode`
- Ett oförändrat objekt ska ge ett oförändrat `hashCode`-värde



### Idén med equals(Object o)

- Ta reda på om två objekt är identiska
  - Varför inte ==?
  - Skillnaden mellan objekt och värden
- Steg för steg
  - Om det är samma objekt är de lika
  - Om de är kompatibla:
    - Gör en explicit typomvandling
    - Kontrollera ifall attributen är identiska
  - Annars är de olika

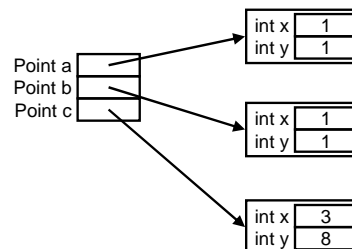


### == och equals

```
int a 12
int b 12
int c 48
```



### == och equals





## equals-metod för Point

```
public class Point {
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (o instanceof Point) {
            Point oPoint = (Point)o;
            return x == oPoint.x && y == oPoint.y;
        } else {
            return false;
        }
    }
}
```



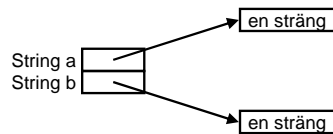
## equals-metod för Point

```
public class Point {
    public boolean equals(Object o) {
        if (this == o) return true;
        if (! (o instanceof Point)) return false;
        Point oPoint = (Point)o;
        return x == oPoint.x && y == oPoint.y;
    }
}
```



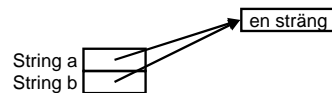
## Men String då?

```
public class Test {
    public static void main(String[] args) {
        String a = new String("en sträng");
        String b = new String("en sträng");
        System.out.println(a == b);
        System.out.println(a.equals(b));
    }
}
```



## Men String då?

```
public class Test {
    public static void main(String[] args) {
        String a = new String("en sträng");
        String b = new String("en sträng");
        System.out.println(a == b);
        System.out.println(a.equals(b));
    }
}
```



## Men String då?

- För att spara plats existerar varje unik sträng bara på ett ställe
- Alla pilar till identiska strängar pekas om så att de pekar ut samma sträng-objekt
- Inga garantier för att det fungerar i framtiden – använd `equals()`!



## Frågor?