Algoritmer och datastrukturer

Binära sökträd
Hash Tabeller

För utveckling av verksamhet, produkter och livskvalitet.

Sökning – *Många datastukturer försöker uppnå den effektivaste sökningen*

- I *arrayer* - linjer sökning, och binärt sökning när arrayen kan vara sörterad
- I *symbol tabeller* , där objekt lagras i par (key och value)
- *Binära Sökträd*
- *Hash tabeller*
- *Binära heap* och *prioritets köer*

För utveckling av verksamhet, produkter och livskvalitet.

Symbol tabeller- med arrayer eller länkade struktur

- För complexa objekt som Customer, Person, mm….och för att kunna uppnå effektiv sökning är det bäst att man associerar en nyckel med ett värde (objekt)

Key	Value

(id_nummer, Customer)

key1	key2	key3		keyN	keyArray
value1	value2	value3		valueN	valueArray

För utveckling av verksamhet, produkter och livskvalitet.

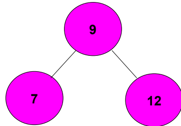
Operationer i en Symbol Tabell -klass

```
public class ST <Key extends Comparable<Key>, Value>
{
ST()  // create a symbol table

 void put(Key key, Value v)  // put key-value pair into the table
 Value get(Key key)          // return value paired with key
 boolean contains(Key key)  // is there a value paired with key?

}
```

För utveckling av verksamhet, produkter och livskvalitet.

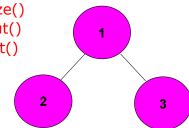
Binär sökträd

- Binära träd
  - Maxim två barn för varje node
    - Left / Right barn
  - Har ordnade element
    - Mindre värde till vänster
    - Högre värde till höger

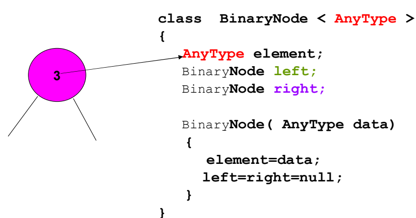


Träd- operationer (metoder)

- Oftast Rekursiva !
  - Mycket effektiva, enkla att förstå
- Vanliga metoder
  - Beräkna antalet noder i trädet, size()
  - Lägg till ny nod i träd, insert(), put()
  - Ta bort nod ur träd, remove(), get()
  - Lista ut innehållet i trädet, print()
    - Pre-Order
    - In-Order
    - Post-Order



classen – Node

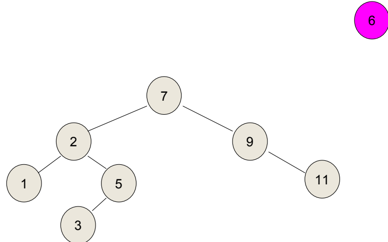



```
class  BinaryNode < AnyType >
{
   AnyType element;
   BinaryNode left;
   BinaryNode right;

   BinaryNode( AnyType data)
    {
      element=data;
     left=right=null;
    }
}
```

...och BinarySearchTree

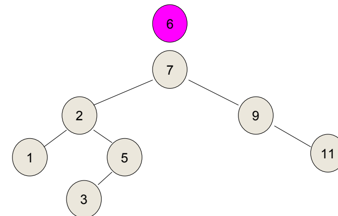
```
public class BinarySearchTree <  AnyType extends

   Comparable<AnyType> >
{
   BinaryNode <AnyType> root;
   public BinarySearchTree() { root= null; }

   -public void insert
   -public nbrOfNodes
   -public remove
   -public find
   .........andra.........
}
```

Insert ()



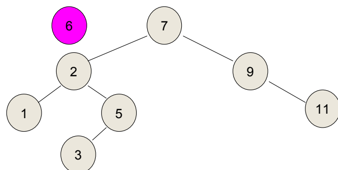
För utveckling av verksamhet, produkter och livskvalitet.

Insert ()



För utveckling av verksamhet, produkter och livskvalitet.

Insert ()



För utveckling av verksamhet, produkter och livskvalitet.

insert(), består av 2 delar

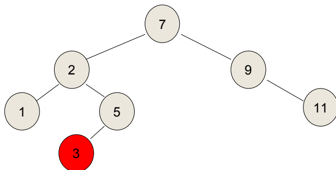
```
public void insert( AnyType x )
      {
       root = findAndInsert( x,  root );

       }
protected BinaryNode<AnyType> findAndInsert( AnyType x,  BinaryNode
<AnyType> t )
      {
         if( t == null )
            t = new BinaryNode <AnyType>( x );
          else if( x.compareTo( t.element ) < 0 )
            t.left = findAndInsert( x, t.left );
          else if( x.compareTo( t.element ) > 0 )
            t.right = findAndInsert( x, t.right );
         else
            throw new DuplicateItemException( x. toString( ) );  // Dubblett ej ok
         return t;
      }
```

anrop av findAndInsert

För utveckling av verksamhet, produkter och livskvalitet.

Remove ()

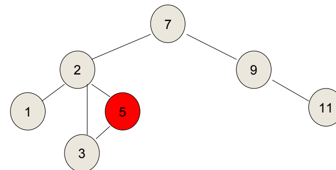
a) Om noden är löv



För utveckling av verksamhet, produkter och livskvalitet.

Remove ()

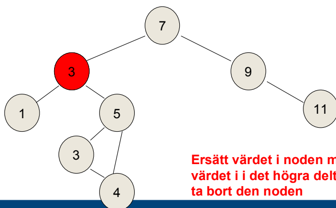
b) Om noden har ett barn



För utveckling av verksamhet, produkter och livskvalitet.

Remove ()

c) Om noden har två barn



Ersätt värdet i noden med den minsta värdet i i det högra delträdet och sedan ta bort den noden

För utveckling av verksamhet, produkter och livskvalitet.

Remove består av 2 delar

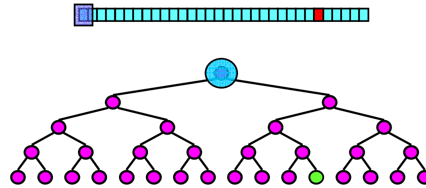
```
public void remove( AnyType x )
    {
        root = findAndRemove( x, root );
    }
```

För utveckling av verksamhet, produkter och livskvalitet.

```
protected BinaryNode<AnyType> findAndRemove( AnyType x,
    BinaryNode<AnyType> t )
      {
        if( t == null )
           throw new ItemNotFoundException( x.toString( ) );
        if( x.compareTo( t.element ) < 0 )
           t.left = findAndRemove( x, t.left );
        else if( x.compareTo( t.element ) > 0 )
           t.right = findAndRemove( x, t.right );
        else if( t.left != null && t.right != null )  // Two children
        {
           t.element = findMin( t.right ).element;
           t.right = findAndRemoveMin( t.right );
        }
        else{
       if( t.left != null )
             t=t.left;    //   only left child
       else
             t=t.right;      ( only right child
      }
        return t;
    }
```

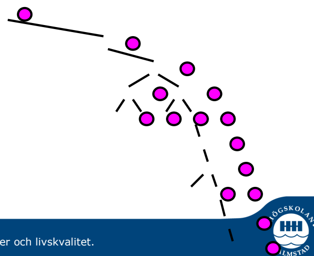
För utveckling av verksamhet, produkter och livskvalitet.

Mycket korta sökningstider!



För utveckling av verksamhet, produkter och livskvalitet.

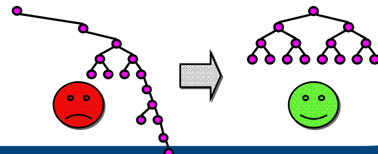
Farliga "worst cases"



För utveckling av verksamhet, produkter och livskvalitet.

Lösningen? Balanserade träd

- Hur ? Försök hålla djupet av trädet så lågt som möjligt.



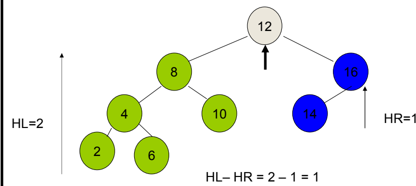
För utveckling av verksamhet, produkter och livskvalitet.

Hur?

- Olika tekniker att balancera
  - AVL Träd
  - Red Black Träd
  - AA Träd
  - Andra ….

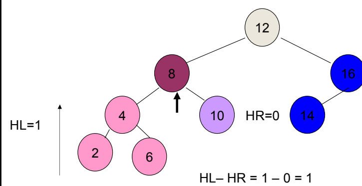
Balanserad SökTräd

Ett träd är balancerad om höjdskillnaden mellan den vänstra delträdet och den högra delträdet är maximum 1.

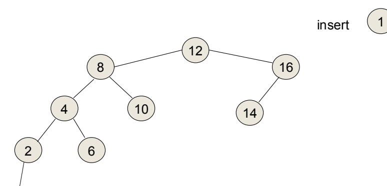


Balanserad Sök Träd – AVL

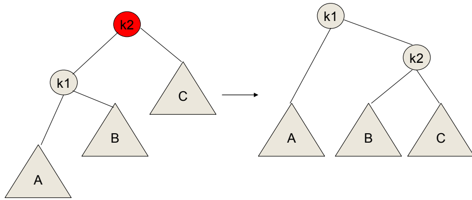
Ett träd är balanserad om höjdskillnaden mellan den vänstra delträdet och den högra delträdet är maximum 1.



BST: AVL – Balanserad?

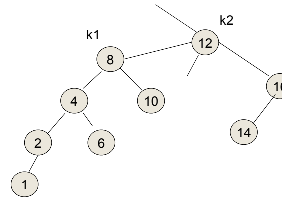


BST: AVL – Rotation höger



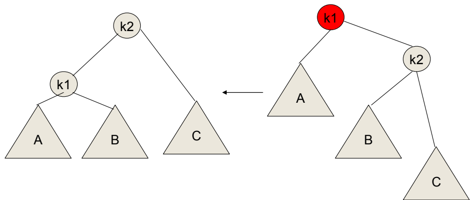
För utveckling av verksamhet, produkter och livskvalitet.

BST: AVL – Höger rotation



För utveckling av verksamhet, produkter och livskvalitet.

BST: AVL – Rotation vänster



För utveckling av verksamhet, produkter och livskvalitet.

Sök träd som Symbol tabell

```
private class Node {
private Key key;                // sorted by key
private Value val;              // associated data private
Node left,
Node right;                     // left and right subtrees private
int N;                          // number of nodes in subtree
public Node(Key key,  Value val, int N)
{
this.key = key;
this.val = val;
this.N = N;
}
}
```

För utveckling av verksamhet, produkter och livskvalitet.

```
public class BST < Key extends Comparable<Key>,  Value>
{
  private Node root; // root of BST
  public void put(Key key, Value val)
  {
    root = put (root, key,  val);
  }
  private Node put(Node x, Key key, Value val)
  {
   if (x == null)
     return new Node(key, val, 1);
    int cmp = key.compareTo(x.key);
   if (cmp < 0)
    x.left = put(x.left, key, val);
    else if ( cmp > 0)
     x.right = put(x.right, key, val);
    else
    x.val = val;   //  if equals, change value in node
    x.N = 1 + size(x.left) + size(x.right);
    return x;
}
```

java.util.*

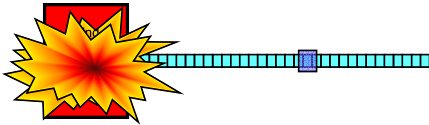
- Innehåller klassen TreeSet och TreeMap som garanterar O(logN) i värsta fall för sökning, insettning och borttagning
- TreeSet- En klass som följer Set-interface (add, contains, remove)
- TreeMap- En klass som följer Map-interface (get, put, remove)
- Båda är implementerade med ett slags balanserad träd (Red-black)
- Objekten som sätts in TreeSet och nycklarna i TreeMap måste implementera interfacen Comparable eller ha en Comparator-objekt.

Hashing / Hash tabeller

Hash Tabell

- Fråga: Varför en annan datastruktur?
- Svar: Konstant tid för både insert () - och find() - operationer

Hash?



För utveckling av verksamhet, produkter och livskvalitet.

Hashfunktioner!

- Hash functioner använder associerade "key" (som kan vara data i sig) för att få fram datan.
- *Hash funktionerna är olika för olika sorts data.*
  - Integers
  - Images
  - Strings

  Etc…

För utveckling av verksamhet, produkter och livskvalitet.

Hashing integers?

- Tänk 16bit int => 0 – 65 535
- Skapa int[] vec = new int[65536];
- Add i => vec[i]++;
- Sök value j => Is vec[j] > 0?
- Ta bort value k => vec[k]--;
- Men för en ... Java int : 32bit
  - 4 miljoner platser => opraktisk!

För utveckling av verksamhet, produkter och livskvalitet.

Exempel av hash funktion

"Daniel"

D	a	n	i	e	l
68	97	110	105	101	108

68 + 97 + 110 + 105 + 101 + 108 = 589

För utveckling av verksamhet, produkter och livskvalitet.

Men...

- Hur unik är hash fuktionen?
  - *hashfunc*(”Daniel”) → 589
  - *hashfunc*(”leinaD”) → 589
- Bättre lösning
  - *bättre hashfunc som skapar mer unika värde* *hash*(”Daniel”) → 129310392
  - *Wrapp* värdet till ett visst intervall
    129310392 % array.lenght (tebellens storlek)

För utveckling av verksamhet, produkter och livskvalitet.

En bättre hash funktion

- Ett bättre sätt att beräkna hash värdet
- Om vi har en text sträng av längd n+1 och alla tecken har index $A_n, A_{n-1},\ldots,A_0$
- gör $s = A_nX^n + A_{n-1}X^{n-1} + \ldots + A_0X^0 =$
- $= ((A_n)X + A_{n-1})X + \ldots + A_0$
- Använd hashValue = s % array.length

För utveckling av verksamhet, produkter och livskvalitet.

Och då......

- Till exempel: ”Danne”

(((’D’)128 + ’a’)128 + ’n’)128 + ’n’)128 + ’e’ =
(((68)128 + 97)128 + 110)128 + 110)128 + 101 =
18 458 851 173

hashValue = 18 458 851 173 % array.length

om length = 7919 (prim nummer) =>

hashValue = 18 458 851 173 % 7919

= 2690

För utveckling av verksamhet, produkter och livskvalitet.

Hur löser man kollision?

- Oavsätt hur unika *nyckel en* hash- funktionen räknar, kollision alltid inträffas.
- Terminology
  - *Load factor*
    - $$LF = \frac{\#\text{used positions}}{\#\text{total avaliable positions}}$$

För utveckling av verksamhet, produkter och livskvalitet.

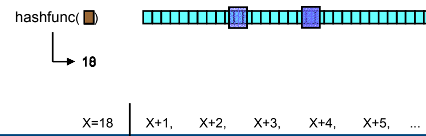
Lösningar

- Linear probing (linjer undersöknings teknik)
- Quadratic probing (kvadratisk undersöknings teknik
- Separate chaining (öppna tabeller)

För utveckling av verksamhet, produkter och livskvalitet.

Linjer undersöknings teknik

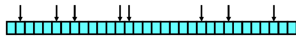
- Sök fram till näst lediga platsen.



För utveckling av verksamhet, produkter och livskvalitet.

Linear probing

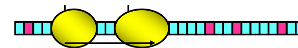
- Fenomen kallat: Primary clusters (primär kluster)



För utveckling av verksamhet, produkter och livskvalitet.

Linjer undersöknings teknik

- Bygger upp kluster
  - Påverkar exekveringstiden för insert () och find() !



För utveckling av verksamhet, produkter och livskvalitet.

Quadratic probing

- Försök undvika "*primary clusters* "
- Snabare en linjär
- Kvadratisk inkrementation av undersökning avståndet

X=18 | X+1^2, X+2^2, X+3^2, X+4^2, X+5^2, ...

Quadratic probing

$3^2 = 9$

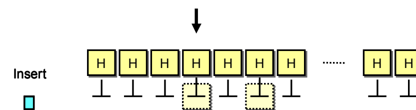
- Garantera att hitta fria platser om de finns

Separate chaining (separat länkning)

- Varje hash position har en länkad lista.
- Påverkar inte andra värde, probing görs bara i listan.
- Varje element I tabellen är en länkad lista.

Separate chaining

Insert



Jämförelse

- Linear probing
  - Enkel
  - Kan resultera i linjär söktid
- Quadratic probing
  - Kräver Load factor < 0.5 annars rehashing ??
  - Kräver prim tal för array storleken
- Separate chaining
  - LF < 1
  - Ingen dubblering, länkade listor är dinamiska !
  - Kan leda till linjär sökning men I verklighetet ganska kort
- Double hash probing
  - Eliminerar kluster

För utveckling av verksamhet, produkter och livskvalitet.