Tvær gagnagrindur sem geyma hluti í röð

- Staflar
  - Bætum stökum “efst” í staflann
  - Tökum stök ofan af staflanum
  - “Last in, first out” – LIFO

- Biðraðir – sleppum biðröðum!
  - Bætum stökum “aftast” í biðröð
  - Tökum stök “fremst” úr biðröðinni
  - “First in, first out” – FIFO

Töflur

Uppflettitafla: lykill og gildi geymt sem par

- Setjum inn lykil með gefið gildi
- Leitum að lykli og finnum gildið

Dæmi: [DNS uppfletting]

- Setjum inn URL fyrir IP tölu
- Gefið URL, finnum IP tölu

URL	IP tala
www.cs.princeton.edu	128.112.136.11
www.princeton.edu	128.112.128.15
www.yale.edu	130.132.143.21
www.harvard.edu	128.103.060.55
www.simpsons.com	209.052.165.60

lykill gildi

Töflur

API fyrir töflur, mjög líkt Map<Key, Value> í java.util

`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`, `null` *if* `key` *not in table*
`boolean`	`contains(Key key)`	*is there a value paired with* `key`*?*

Note: Implementations should also implement the `Iterable<Key>` *interface to enable clients to access keys in sorted order with foreach loops.*

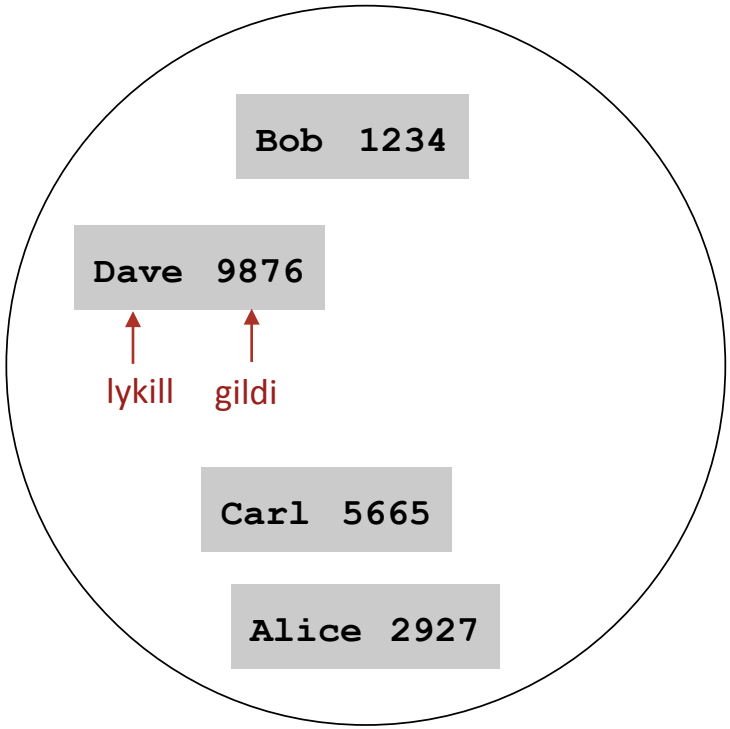
Töflur

```
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`, `null` *if* `key` *not in table*
`boolean`	`contains(Key key)`	*is there a value paired with* `key`*?*

Note: Implementations should also implement the `Iterable<Key>` *interface to enable clients to access keys in sorted order with foreach loops.*

Tafla geymir safn lykla-gilda para

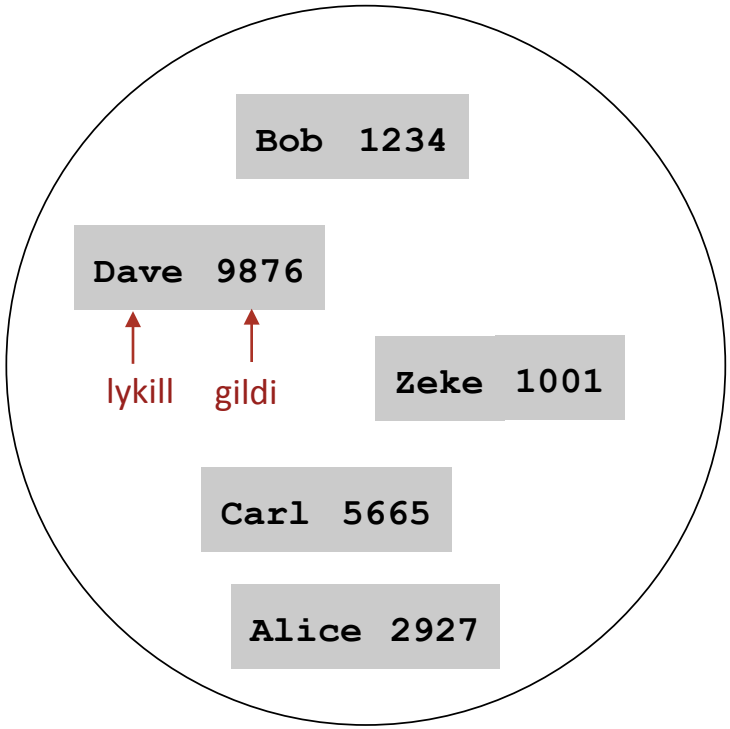


Töflur

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, null *if* key *not in table*
boolean contains(Key key)	*is there a value paired with* key*?*

Note: Implementations should also implement the Iterable<Key> *interface to enable clients to access keys in sorted order with foreach loops.*

put("Zeke", 1001)
bætir við töfluna

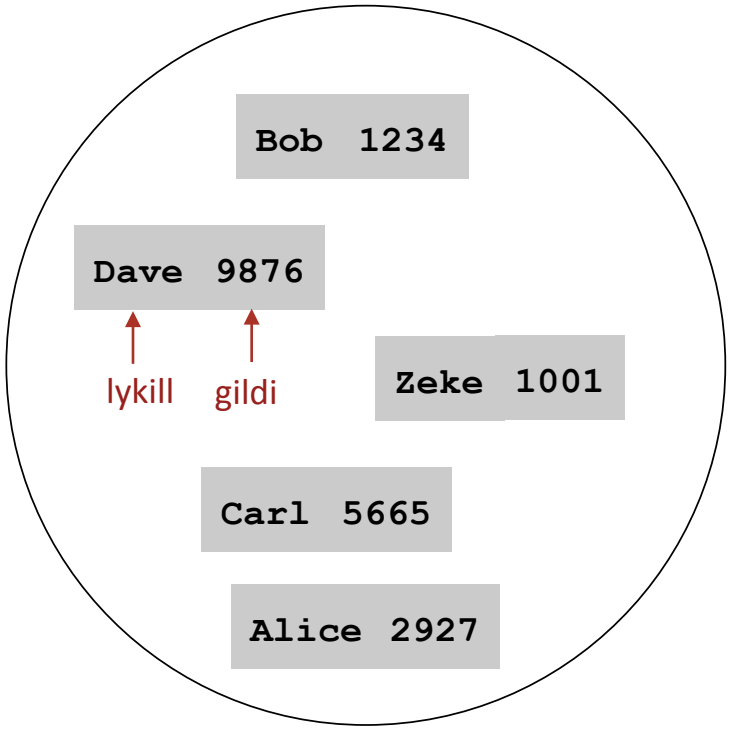


Töflur

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, null *if* key *not in table*
boolean contains(Key key)	*is there a value paired with* key*?*

Note: Implementations should also implement the Iterable<Key> *interface to enable clients to access keys in sorted order with foreach loops.*

get("Alice")
skilar 2927



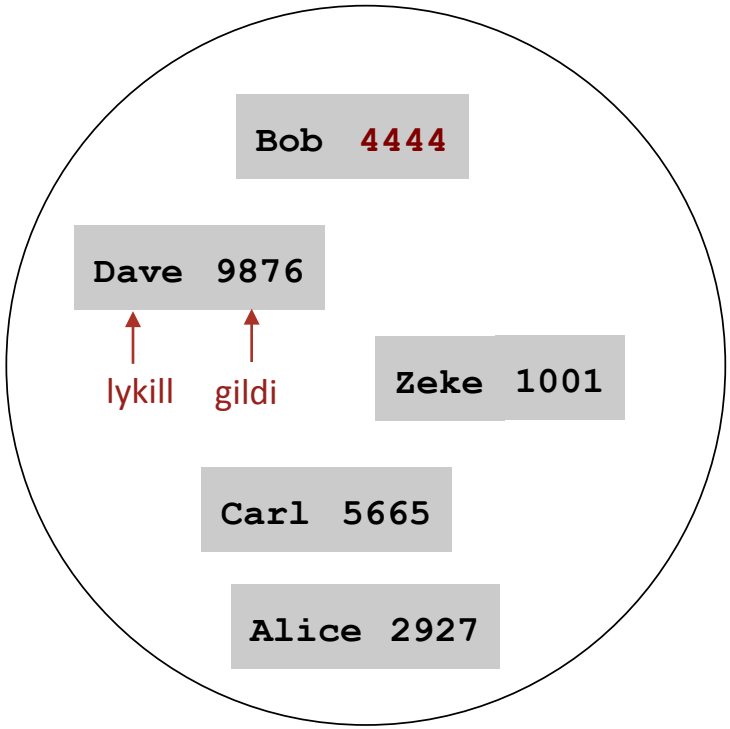
Töflur

```
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`, `null` *if* `key` *not in table*
`boolean`	`contains(Key key)`	*is there a value paired with* `key`*?*

Note: Implementations should also implement the `Iterable<Key>` *interface to enable clients to access keys in sorted order with foreach loops.*

put("Bob",4444)
breytir gildinu
fyrir "Bob"



Tíðnitölur

Mælum tíðni [t.d. netumferð, orðaföldi]

- Lesum inn lykil
- Ef lykillinn er í töflunni hækkum töluna um 1
- Ef lykillinn er ekki í töflunni, bætum við með gildið 1

```
public class Freq {
   public static void main(String[] args) {
      ST<String, Integer> st = new ST<String, Integer>();
      while (!StdIn.isEmpty()) {
         String key = StdIn.readString();
         if (st.contains(key)) st.put(key, st.get(key) + 1);
         else                  st.put(key, 1);
      }
      for (String s : st)
         StdOut.println(st.get(s) + " " + s);
   }
}
```

lykla tag

gildis tag

reikna tíðni

foreach lykkja, í næstu viku!

prenta tíðni

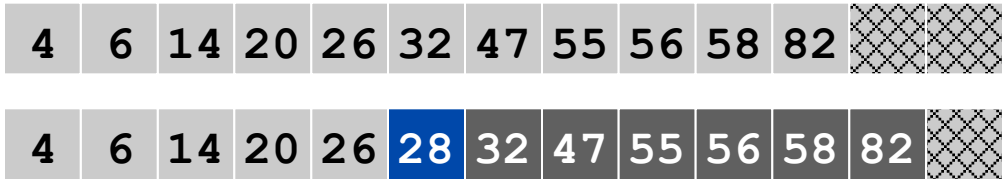
Töflur – einfaldar útfærslur

Óraðað fylki

- Put: bætum lyklinum og gildinu aftast í lista
- Get: leitum að lyklinum í fylkinu

Raðað fylki

- Put: finnum réttan stað fyrir lykilinn skv. röð, hliðrum til að búa til pláss
- Get: helmingunarleit til að finna lykil



eftir innsetningu 28

Töflur – einfaldar útfærslur

Óraðað fylki: vonlaust fyrir stór fylki

Raðað fylki: slæmt fyrir margar innsetningar, ásættanlegt ef það er kallað mun oftar á get()

implementation	get	put	Moby	100K	200K	1M
unordered array	N	N	170 sec	4.1 hr	-	-
ordered array	$\log N$	N	5.8 sec	5.8 min	15 min	2.1 hr

Viljum fá log(N) fyrir bæði get og put

Enn betra O(1) fyrir bæði get og put!

Töflur

- Óröðuð fylki: put() var hægt því að við þurftum fyrst að leita fyrst og svo að setja aftast í fylki (fljót aðgerð)

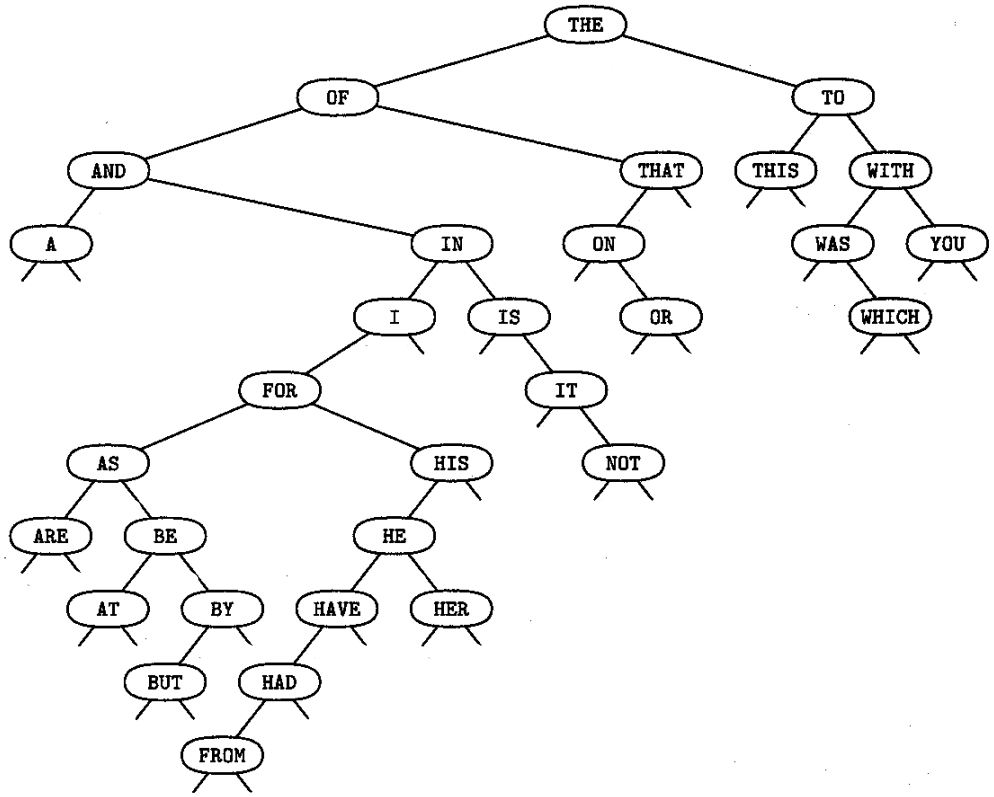
- Röðuð fylki: helmingunarleitin er hraðvirk, put() er of hægt því það er erfitt að setja inn í mitt fylkið

- Tengdir listar: auðvelt að setja inn í miðjan lista, erfitt að leita (tekur O(N) tíma að finna stakið í miðjunni)

Viljum milliveg: auðvelta að leita og
auðvelt að setja inn í “miðjuna”

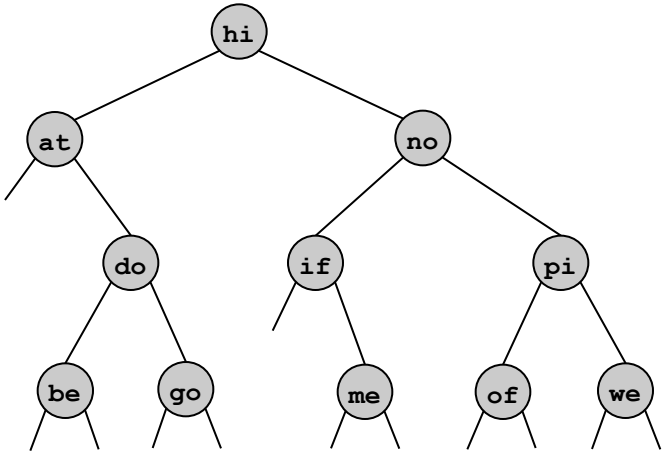
Töflur - tvíundartré

Lausnin er að nota tvíundartré



Töflur - tvíundartré

Tvíleitartré (Binary Search Tree, BST) er tvíundartré sem varðveitir röðun

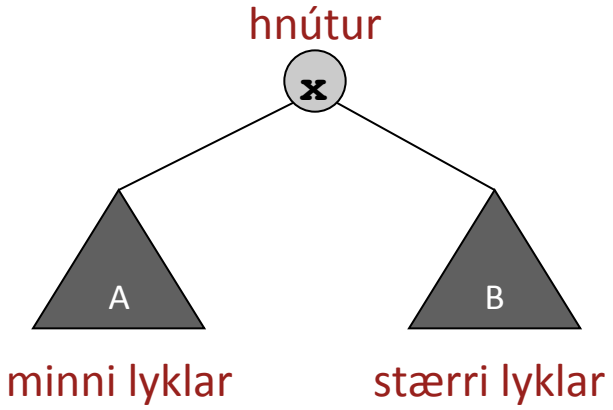


Tvíundartré er annað hvort

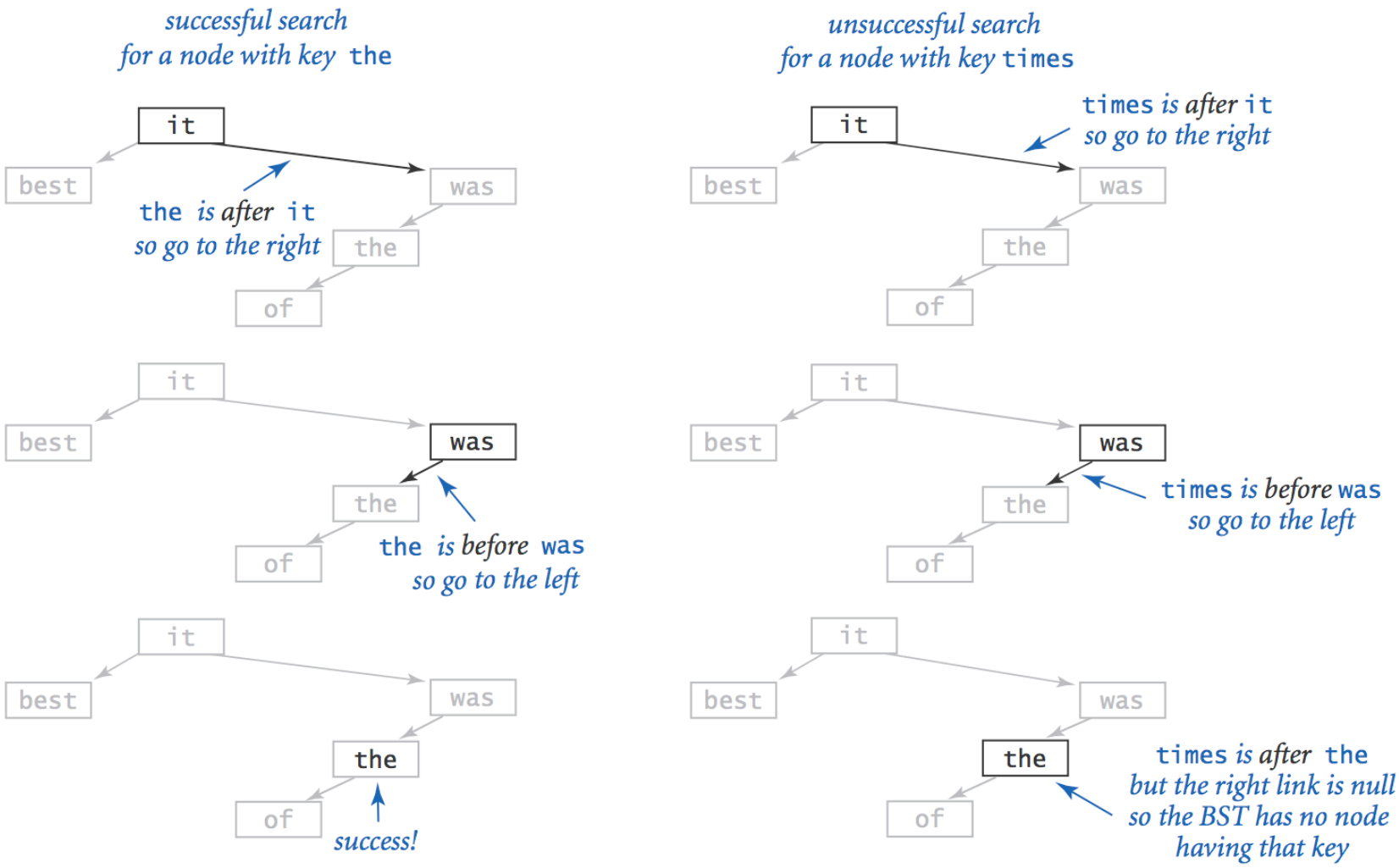
- tómt
- (lykill,gildi) og tvö tvíundartré, vinstra og hægra tré (börnin)

Varðveitir röðun:

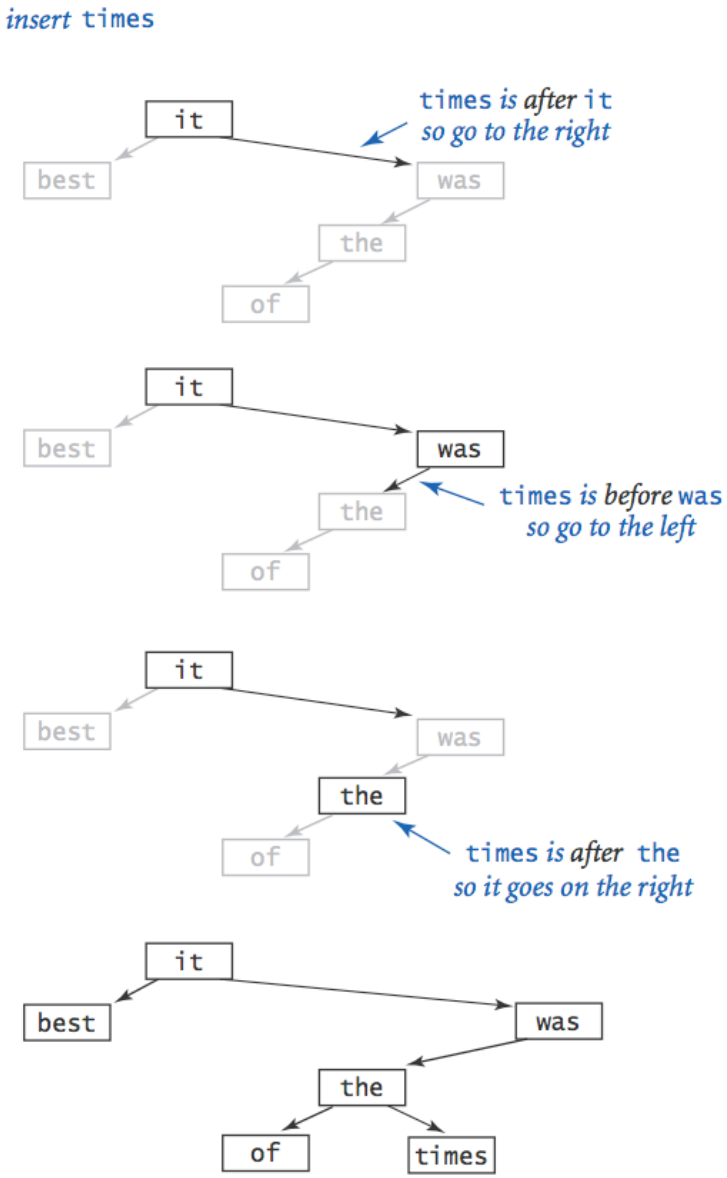
- Allir lyklar í vinstra tré eru minni en lykill foreldris
- Allir lyklar í hægra tré eru stærri en lykill foreldris



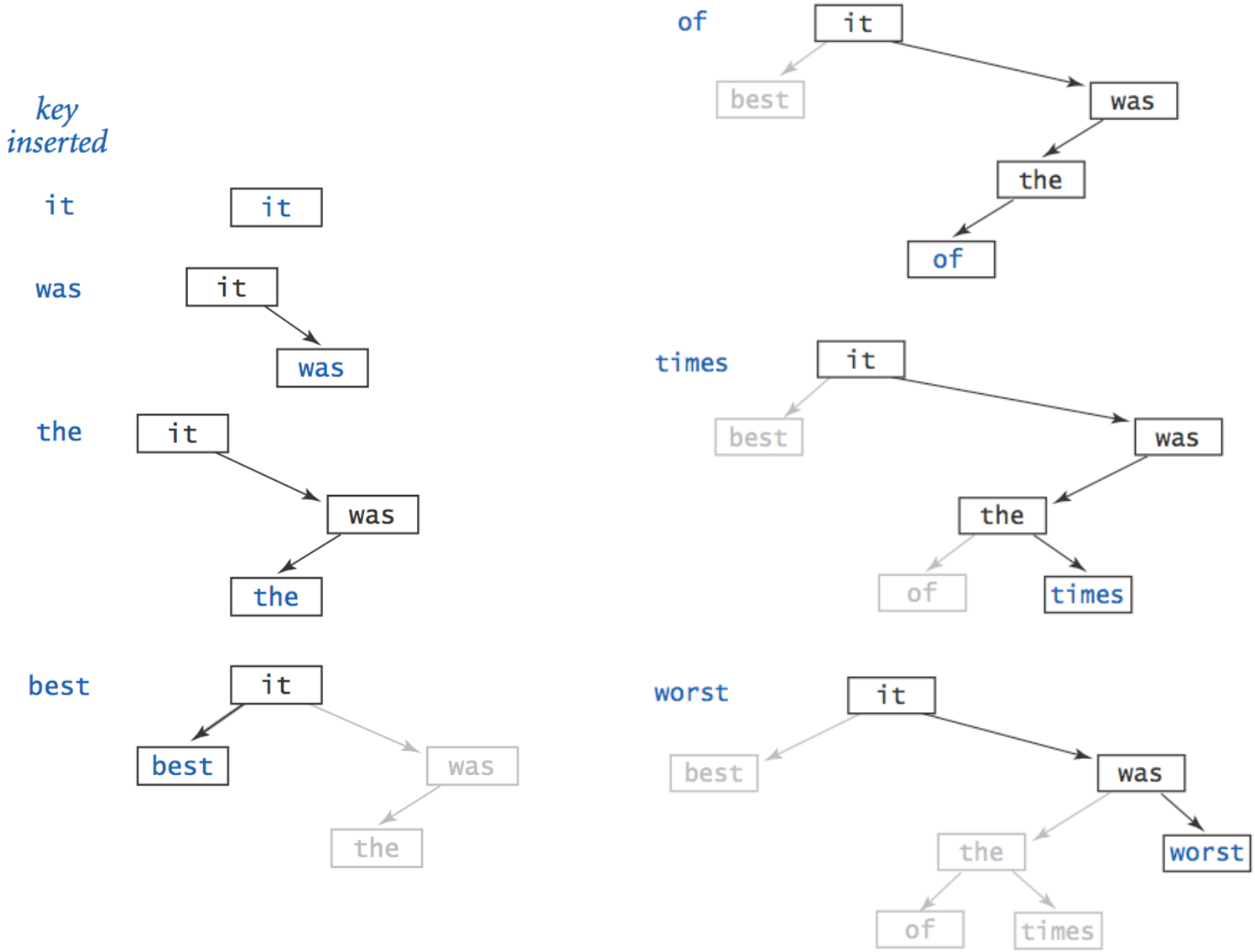
BST leit


successful search
for a node with key the
it
best
was
the
of
the is after it
so go to the right
the is before was
so go to the left
success!
unsuccessful search
for a node with key times
times is after it
so go to the right
times is before was
so go to the left
times is after the
but the right link is null
so the BST has no node
having that key

BST innsetning



BST vöxtur


key inserted
it
it
was
it
was
the
it
was
the
best
it
best
was
the
of
it
best
was
the
of
times
it
best
was
the
of
times
worst
it
best
was
the
worst
of
times

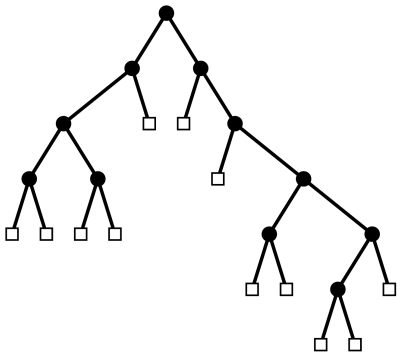
BST útfærsla

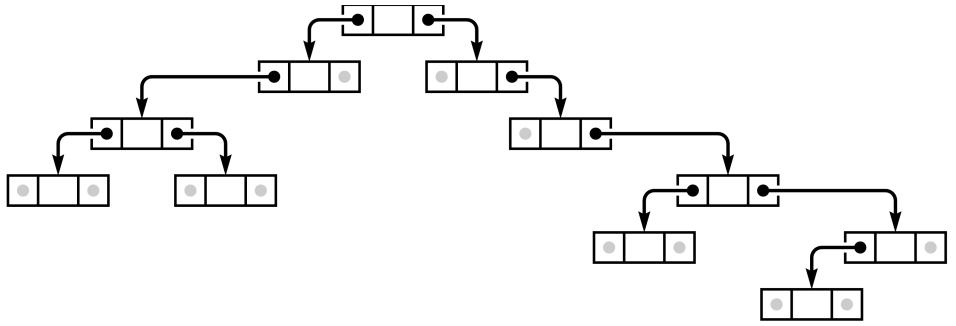
Notum tvær tilvísanir í hverjum hnút

Hver hnútur hefur

- lykil
- gildi
- tilvísun á vinstra tré
- tilvísun á hægra tré

```
private class Node {
   private Key key;
   private Value val;
   private Node left;
   private Node right;
}
```





BST útfærsla

Upphafsstillt sem null

```
public class BST<Key extends Comparable<Key>, Value> {

   private Node root;   // root of the BST

   private class Node {
      private Key key;
      private Value val;
      private Node left, right;

      private Node(Key key, Value val) {
         this.key = key;
         this.val = val;
      }
   }

   public void put(Key key, Value val)  { … }
   public Value get(Key key)            { … }
   public boolean contains(Key key)     { … }

}
```

BST get

Get: skilar gildinu `val` fyrir lykilinn `key` eða `null` ef hann finnst ekki

```
public Value get(Key key) {
   return get(root, key);
}

private Value get(Node x, Key key) {
   if (x == null) return null;
   int cmp = key.compareTo(x.key);
   if      (cmp < 0) return get(x.left,  key);
   else if (cmp > 0) return get(x.right, key);
   else              return x.val;
}

public boolean contains(Key key) {
   return (get(key) != null);
}
```

neikvætt ef minni,
jákvætt ef stærri

BST put

Put: látum lykilinn `key` fá gildið `val`

- leita fyrst, setja svo inn í tré
- stuttur, en lúmskur, endurkvæmur kóði

```
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);
   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;
   return x;
}
```

Skrifum yfir gamla gildið