Sökträd: Splay-träd, prio-köer, heapar

TDDE22, 725G97: DALG

Magnus Nielsen

LINKÖPING UNIVERSITY

LINKÖPING UNIVERSITY

Binära sökträd är inte unika

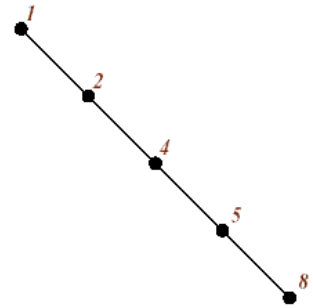
Kom ihåg det binära sökträdet:

- Enkelt att sätta in och ta bort element, men...
- "balansen" bestäms av ordningen på insättningar och borttagningar.

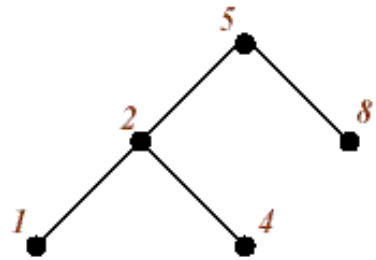
Kombinera med heuristiken "håll nyligen använda element först" för listor?

- Ofta använda element bör finnas nära roten!

insert: 1,2,4,5,8



insert: 5,2,1,4,8

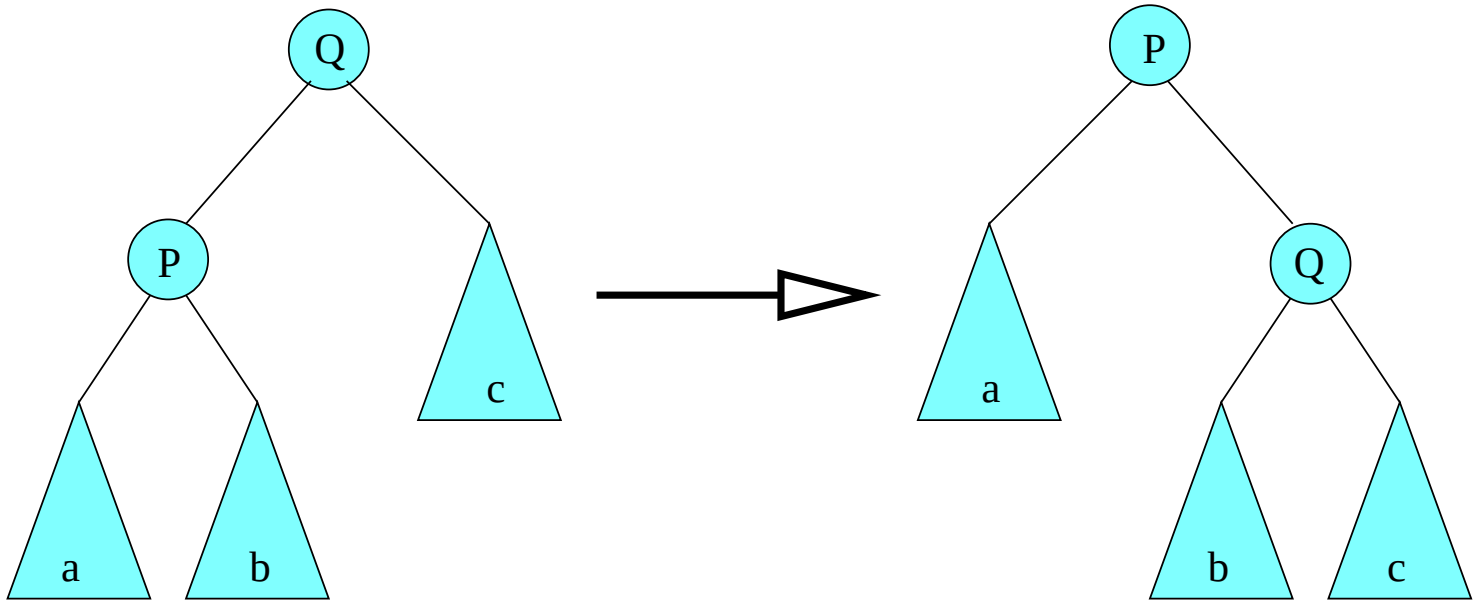


Operationen splay(k)

- Utför en normal sökning efter k, kom ihåg noderna vi passerar...
- Märk den sista noden vi undersöker med P
  - Om k finns i T, finns k i noden P,
  - annars är P förälder till ett tomt träd
- Återvänd till roten och gör en rotation vid varje nod för att flytta P uppåt i trädet...(3 fall)

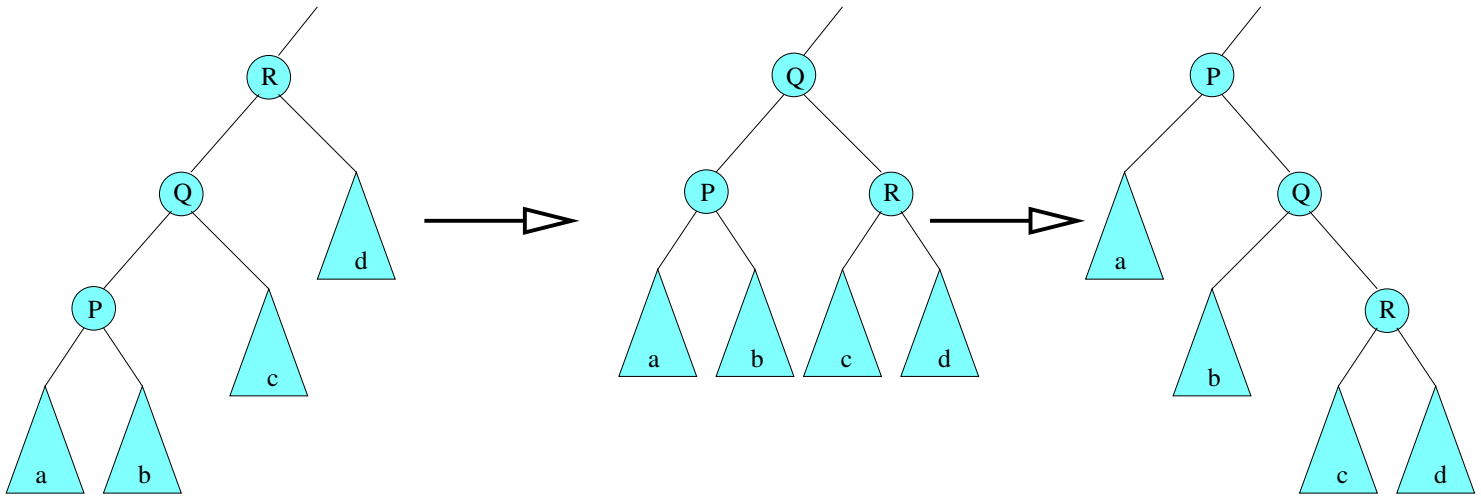
Operationen splay(k)

- zig: parent(P) är roten: rotera kring P



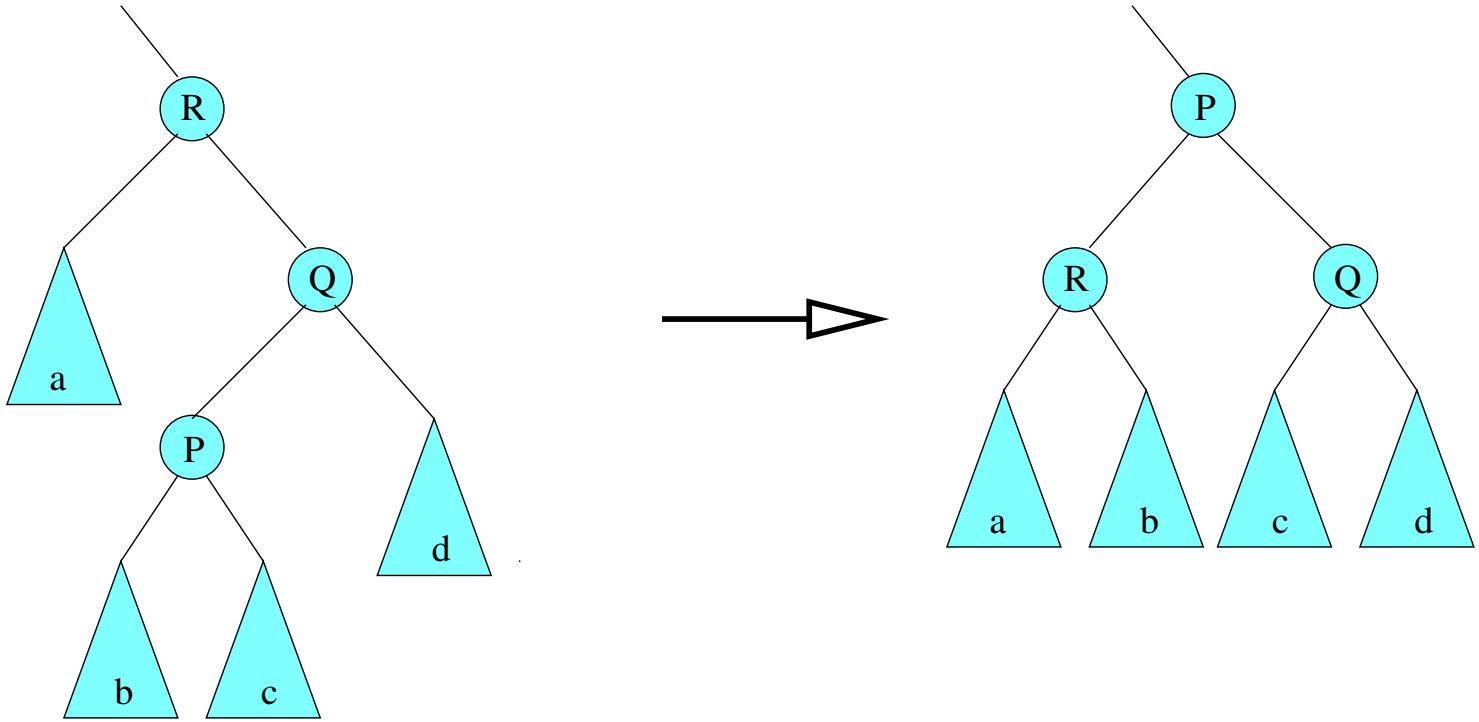
Operationen splay(k)

- zig-zig: P och parent(P) är bägge vänsterbarn (eller bägge högerbarn): utför två rotationer för att flytta upp P



Operationen splay(k)

- zig-zag: En av P och parent(P) är ett vänsterbarn och den andra är ett högerbarn eller vice versa: utför två rotationer i olika riktningar

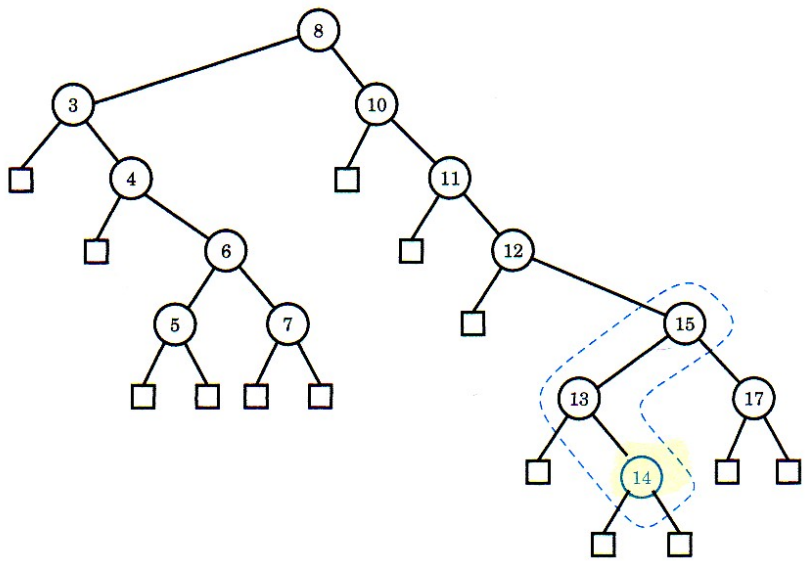


find och insert

function FIND(k, T)
    SPLAY(k, T)
    if KEY(ROOT$(T)) = k$ **then return** (k, v)
    else return null

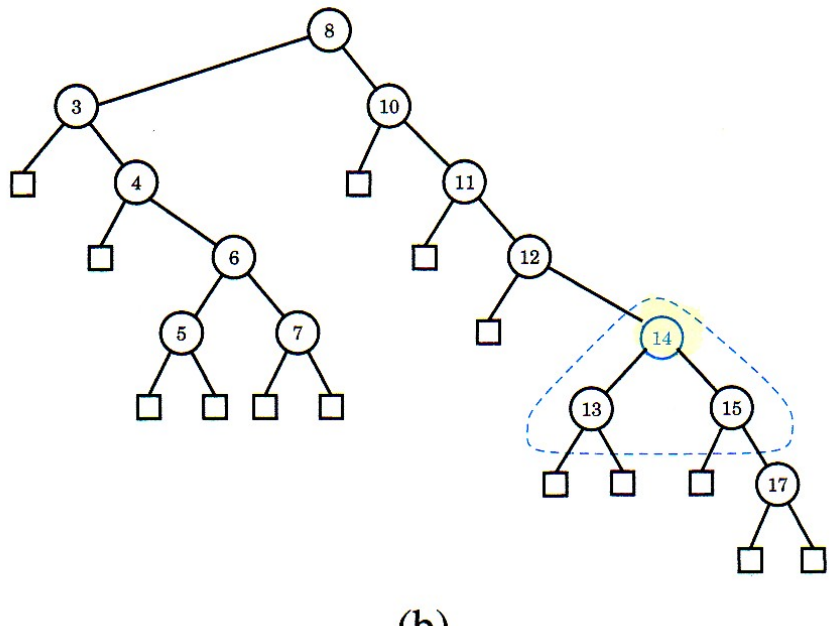
function INSERT(k, v, T)
    sätt in (k, v) som i ett binärt sökträd
    SPLAY(k, T)

Exempel: insättning av 14



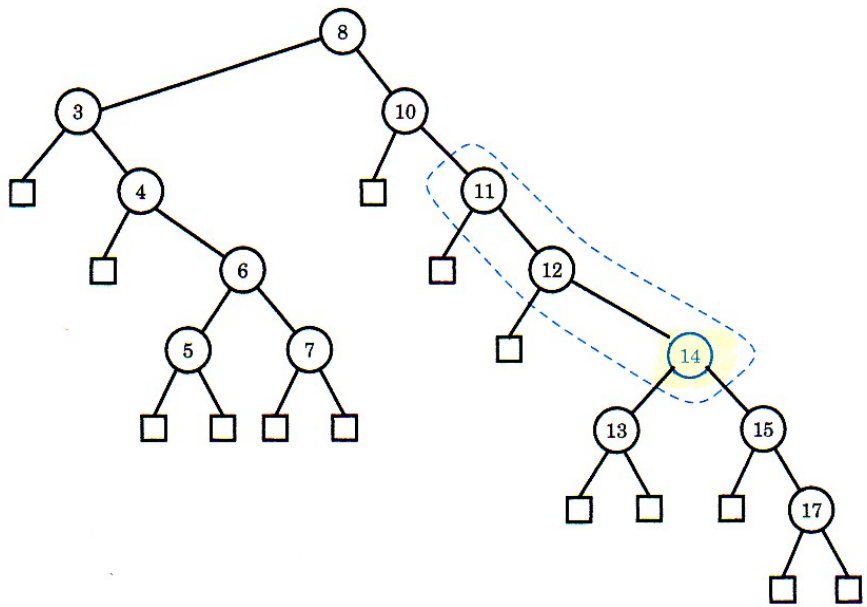
(a)

Exempel: insättning av 14

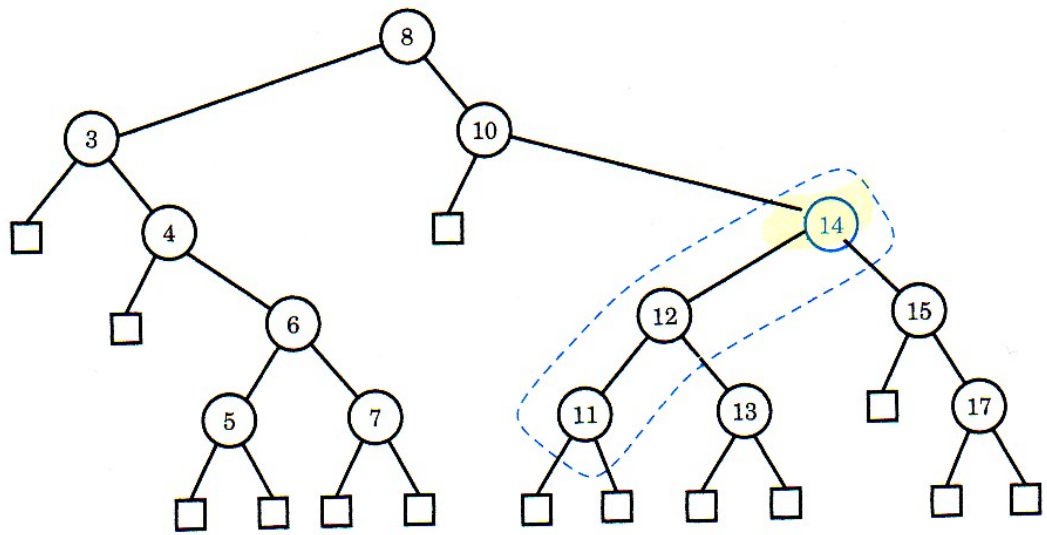


(b)

Exempel: insättning av 14

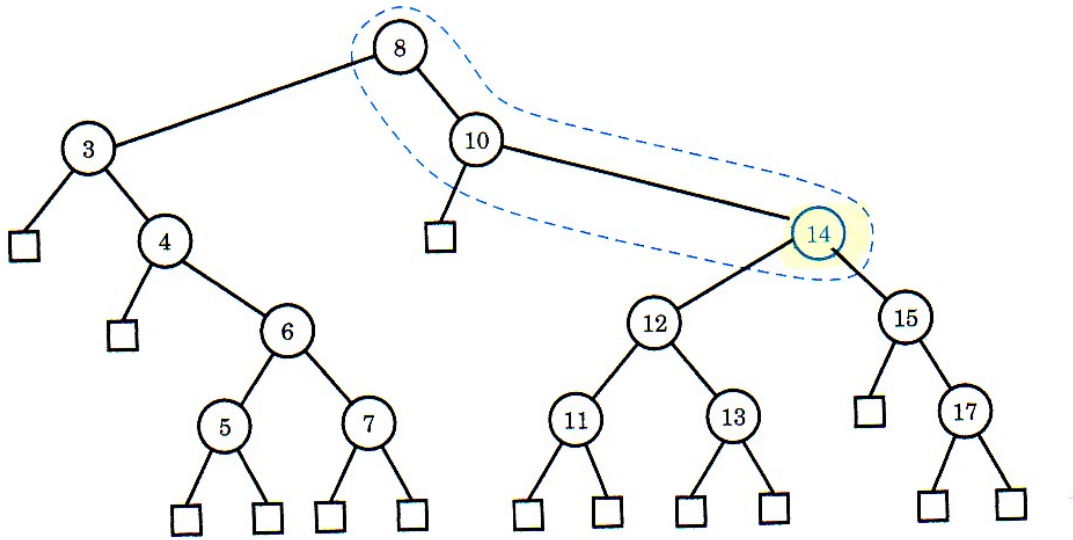


Exempel: insättning av 14



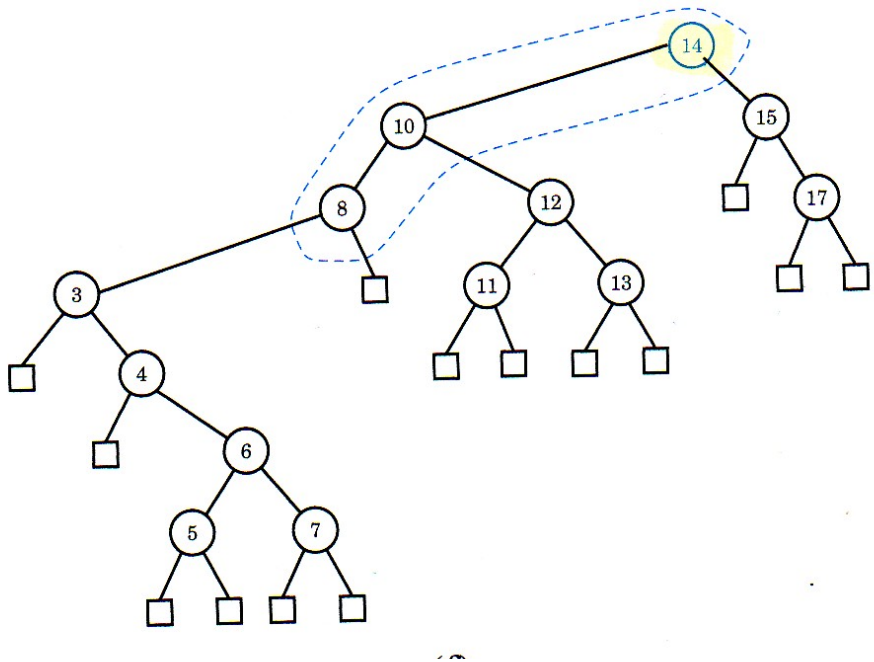
(d)

Exempel: insättning av 14



(e)

Exempel: insättning av 14

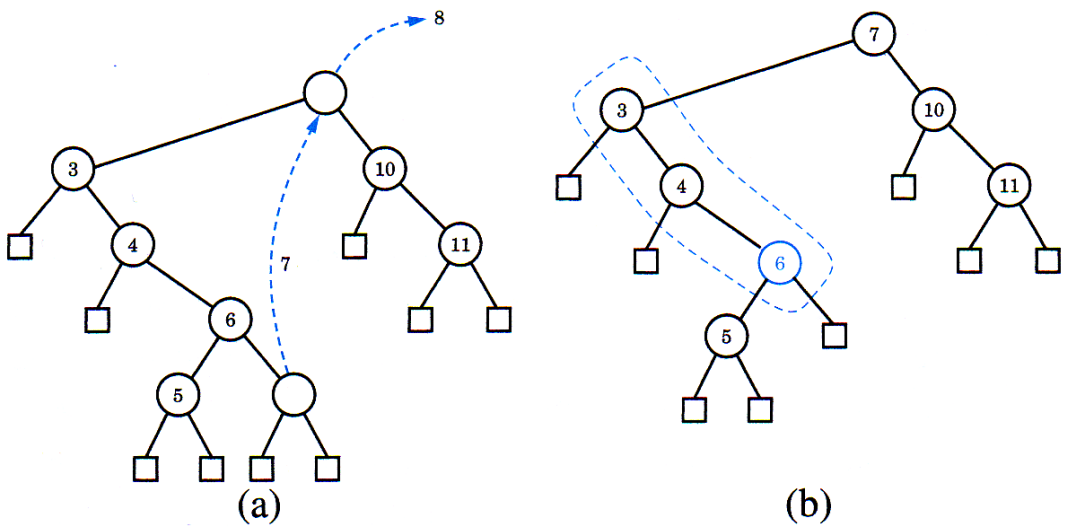


delete

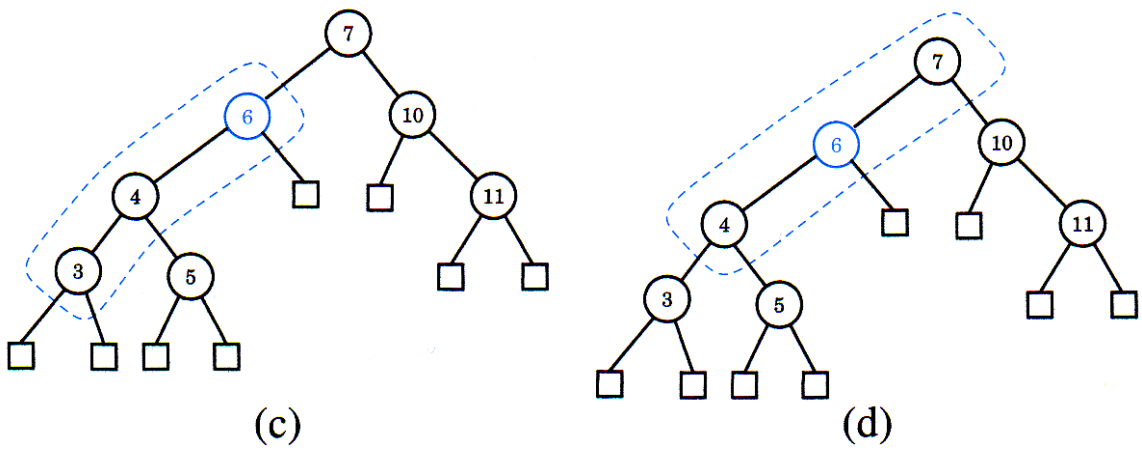
function DELETE(k, T)
  if k finns i ett löv **then**
    gör SPLAY på föräldern till lövet
  else if k finns i en intern nod **then**
    ersätt noden med dess föregångare i inorder
    gör SPLAY på föräldern till föregångaren

Det går förstås att använda efterföljaren i inorder också.

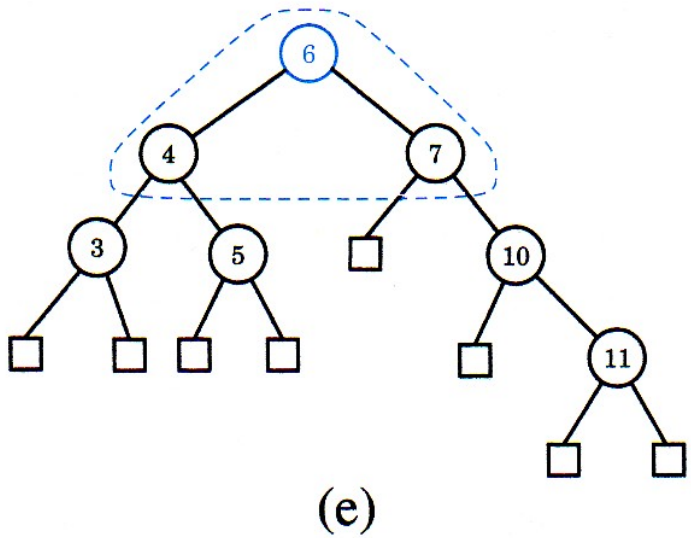
Exempel: borttagning av 8



Exempel: borttagning av 8



Exempel: borttagning av 8



Användning

- Akademiskt intresse (DALG)
- Vissa (i synnerhet äldre) routers

Prestanda

- Varje operation kan behöva utföras på ett totalt obalanserat träd
  - alltså ingen garanti för tid $O(\log n)$ i värsta fallet

- Amorterade tiden är logaritmisk
  - varje sekvens av m operationer, utförda på ett initialt tomt träd, tar totalt $O(m \log m)$ tid
  - alltså är den *amorterade* kostnaden/tiden för en operation $O(\log n)$ även om enskilda operationer kan bete sig mycket värre

1 Splay-träd
2 Prioritetsköer
3 Heapar

LINKÖPING UNIVERSITY

Prioritetsköer

En vanligt förekommande situation:

- Väntelista (jobbhantering på flera användardatorer, simulering av händelser)
- Om en resurs blir ledig, välj ett element från väntelistan
- Valet är baserat på någon partial/linjär ordning:
  - jobbet med högst prioritet ska köras först,
  - varje händelse ska inträffa vid en viss tidpunkt; händelserna ska bearbetas i tidsordning

ADT prioritetskö

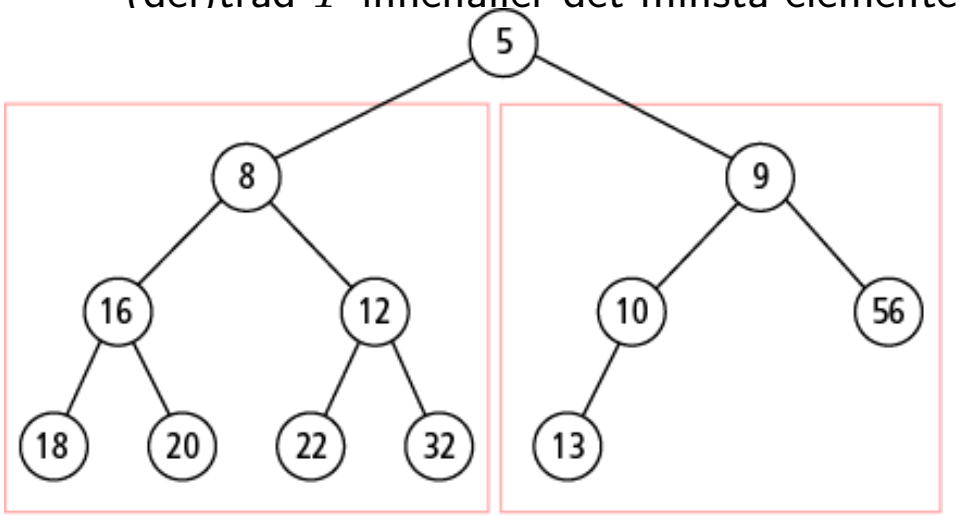
- Linjärt ordnad mängd K av nycklar
- Vi lagrar par (k, v) (som i ADT Dictionary), *flera par* med *samma nyckel* är tillåtet
- en vanlig operation är att hämta par med minimal nyckel
- Operationer på en prioritetskö P:
  - makeEmptyPQ()
  - isEmpty()
  - size()
  - min(): hitta ett par (k, v) som har minimalt k i P; returnera (k, v)
  - insert(k, v): sätt in (k, v) i P
  - removeMin(): ta bort och returnera ett par (k, v) i P med minimalt k; **error** om P är tom

Implementation av prioritetsköer

- Vi kan t.ex. använda (sorterade) länkade listor, BST eller Skip-listor

Implementation av prioritetsköer

- Vi kan t.ex. använda (sorterade) länkade listor, BST eller Skip-listor
- En annan idé: använd ett komplett binärt träd där roten i varje (del)träd T innehåller det minsta elementet i T



Det här är ett partiellt ordnat träd, också kallat heap!

Ett komplett binärt träd i naturen

LINKÖPING UNIVERSITY

Att uppdatera en heapstruktur

- Med sista lövet menar vi den sista noden i en traversering i levelorder

- removeMin(PQ) // ta bort roten
  - Ersätt roten med sista lövet
  - Återställ den partiella ordningen genom att byta noder nedåt "down-heap bubbling"

- insert(PQ, k, v)
  - Sätt in ny nod (k, v) efter sista lövet
  - Återställ den partiella ordningen genom "up-heap bubbling"

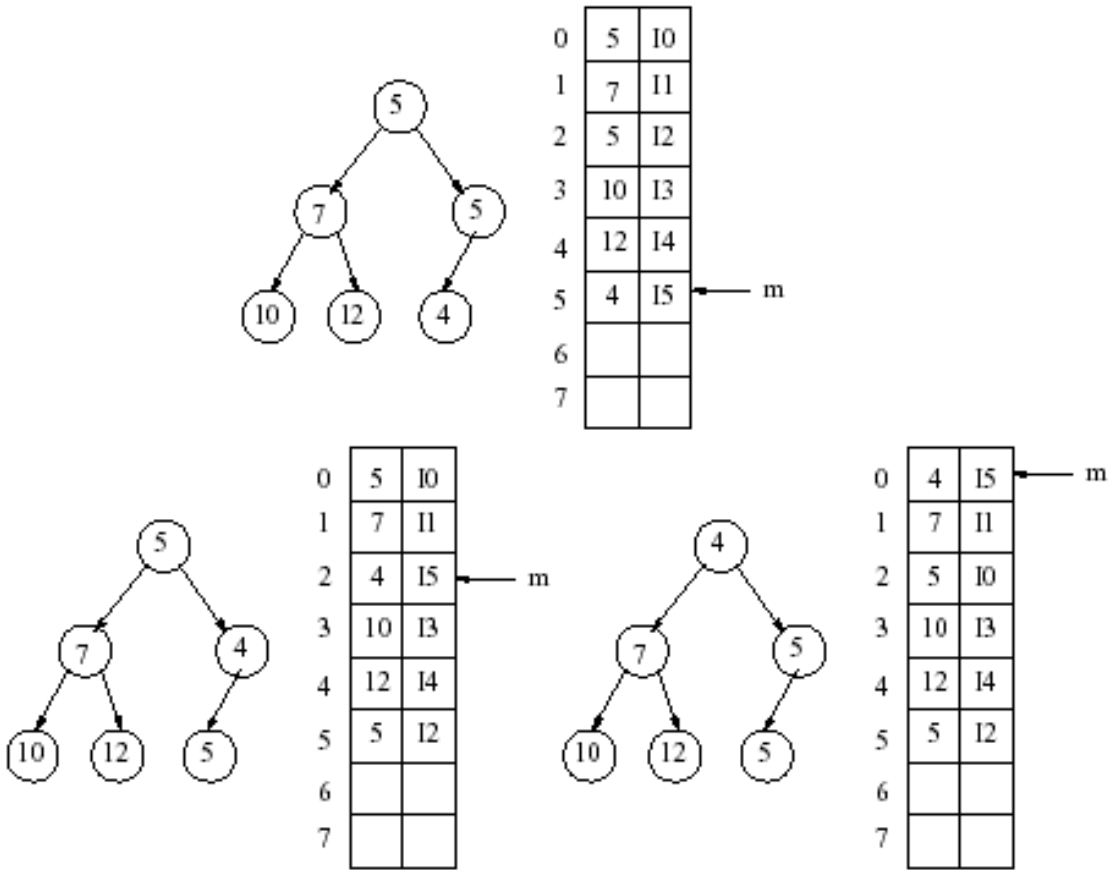
Egenskaper

- size(), isEmpty(), min(): $O(1)$
- insert(), removeMin(): $O(\log n)$

Kom ihåg arrayrepresentationen av BST
Ett komplett binärt träd...

- Kompakt arrayrepresentation
- ”Bubble-up” och ”bubble-down” har snabba implementationer

Exempel: ”bubble-up” efter insert(4,15)



Heapvarianter

Olika partialordningar

- minsta nyckeln i roten (minHeap)
- största nyckeln i roten (maxHeap)

Nästa gång:
Introduktion till sortering...

www.liu.se

LINKÖPING UNIVERSITY