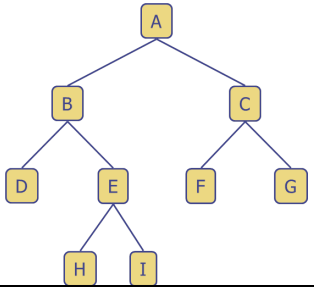
Binarna Stabla

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2022.

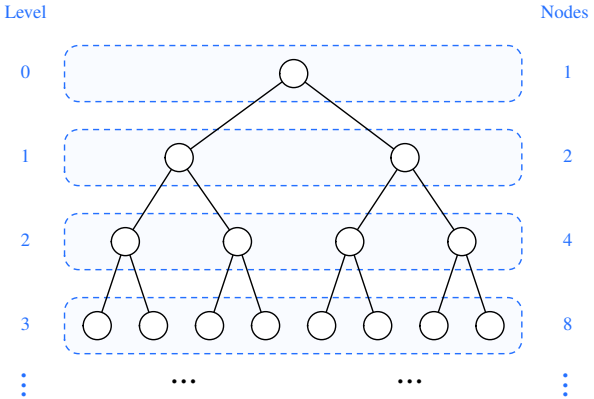
Binarno stablo

- stablo za koje važi:
  - svaki čvor ima najviše dvoje dece
  - svako dete je označeno kao levo dete ili desno dete
  - levo dete po redosledu prethodi desnom detetu
- levo podstablo – levo dete kao koren
- desno podstablo – desno dete kao koren
- pravilno binarno stablo: svaki čvor ima 0 ili 2 deteta

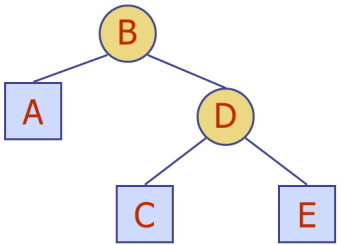


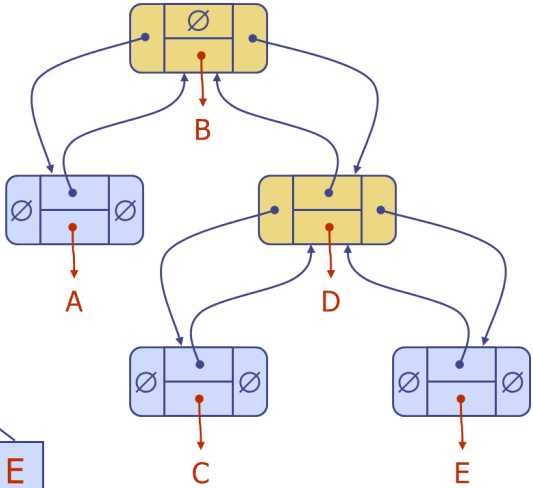
Osobine binarnog stabla

- nivo stabla d ima najviše 2^d čvorova
- broj čvorova po nivou raste eksponencijalno



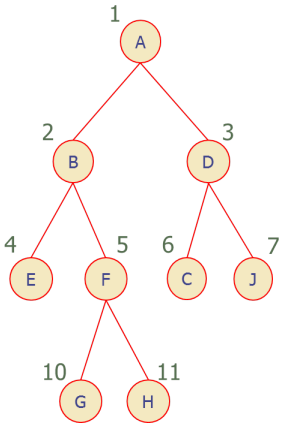
Binarno stablo u memoriji / čvorovi i reference





Binarno stablo u memoriji / pomoću niza

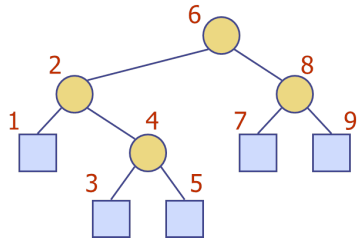
- rang čvora:
  - rang$(root) = 1$
  - za levo dete:
    rang$(node) = 2 \cdot$ rang$(parent)$
  - za desno dete:
    rang$(node) = 2 \cdot$ rang$(parent) + 1$
- čvor v se smešta u $A[\text{rang}(v)]$



Obilazak binarnog stabla / inorder

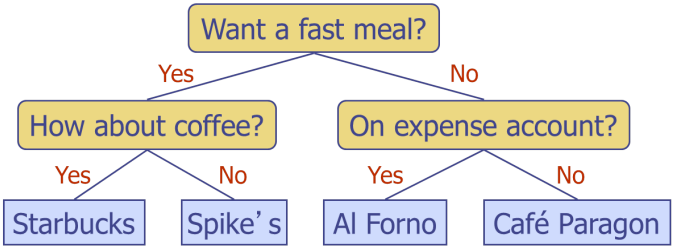
inorder(n)

if n ima levo dete **then**
  inorder(levo dete)
obradi(n)
if n ima desno dete **then**
  inorder(desno dete)



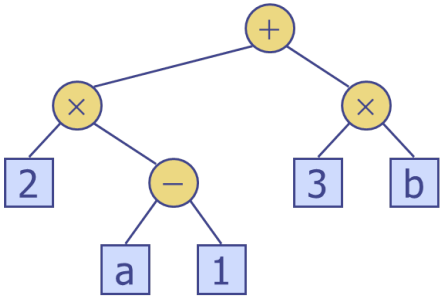
Stabla odlučivanja

- binarno stablo strukturirano prema procesu odlučivanja
- unutrašnji čvorovi – pitanja sa da/ne odgovorima
- listovi – odluke
- primer: gde za večeru?



Stablo aritmetičkih izraza

- binarno stablo kreirano na osnovu aritmetičkog izraza
- unutrašnji čvorovi – operatori
- listovi – operandi
- primer: $2 * (a - 1) + 3 * b$

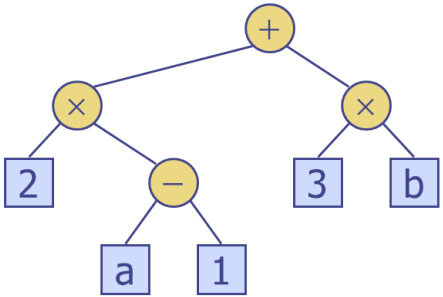


Ispisivanje aritmetičkih izraza

- specijalni slučaj **inorder** obilaska

printExpr(n)

if n ima levo dete **then**
  print("(")
  printExpr(levo dete)
print(n)
if n ima desno dete **then**
  printExpr(desno dete)
  print(")")

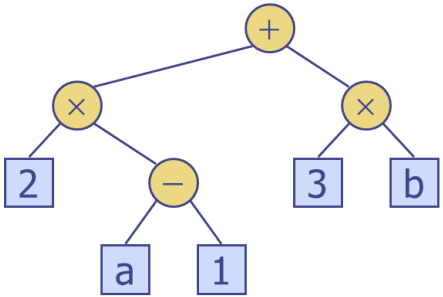


Izračunavanje aritmetičkih izraza

- specijalni slučaj **postorder** obilaska

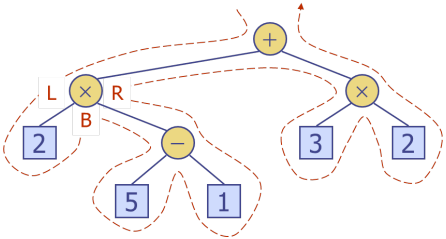
evalExpr(n)

if n je list **then**
  return $n.element$
else
  $x \leftarrow evalExpr(n.left)$
  $y \leftarrow evalExpr(n.right)$
  $\diamond \leftarrow$ operator u n
  return $x \diamond y$



Ojlerov obilazak stabla

- opšti postupak za obilazak stabla
- preorder, inorder, postorder su specijalni slučajevi
- posmatramo grane stabla kao zidove koji uvek moraju da nam budu sa leve strane prilikom kretanja
- svaki čvor se poseti tri puta
  - sa leve strane (preorder)
  - sa donje strane (inorder)
  - sa desne strane (postorder)



Mape sa poretkom

- postoji relacija poretka nad ključevima
- elementi se skladište prema vrednosti ključa
- pretrage „najbliži sused“ (nearest neighbor):
  - nađi element sa najvećim ključem manjim ili jednakim k
  - nađi element sa najmanjim ključem većim ili jednakim k

Binarna pretraga

- binarna pretraga može da pronađe „najbližeg suseda“ za mapu sa poretkom implementiranu pomoću niza koji je sortiran po ključu
  - u svakom koraku prepolovi se broj kandidata
  - radi u $O(\log n)$ vremenu
- primer: nađi 7

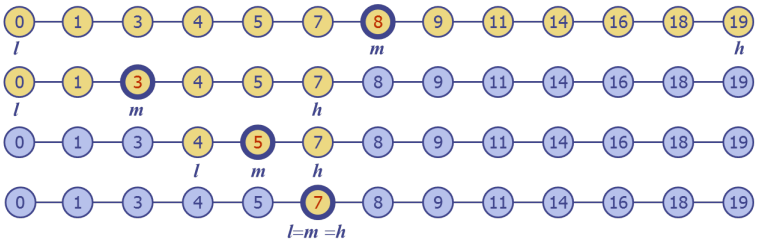


Tabela pretrage

- tabela pretrage je mapa sa poretkom implementirana pomoću sortiranog niza
  - eksterni komparator za ključeve
- performanse:
  - binarna pretraga je $O(\log n)$
  - dodavanje je $O(n)$
  - uklanjanje je $O(n)$
- radi efikasno samo za mali broj elemenata ili tamo gde je pretraga česta a izmene retke (npr. provera kreditne kartice)

Sortirana mapa ATP

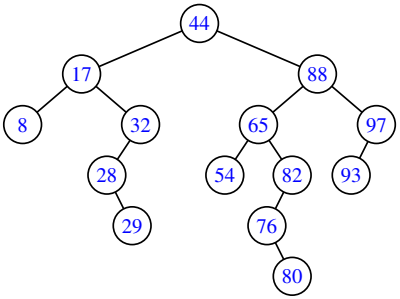
- standardne operacije mape

`M[k]`	vraća vrednost v za ključ k u mapi M; implementira je `__getitem__`
`M[k]=v`	dodaje novi element (k, v) u M ili menja postojeći; implementira je `__setitem__`
`del M[k]`	uklanja element sa ključem k iz M; implementira je `__delitem__`

- dodatne funkcionalnosti
  - sortiran redosled prilikom iteracije
  - nađi veće: find_gt(k)
  - nađi u opsegu: find_range($start, stop$)

Binarno stablo pretrage

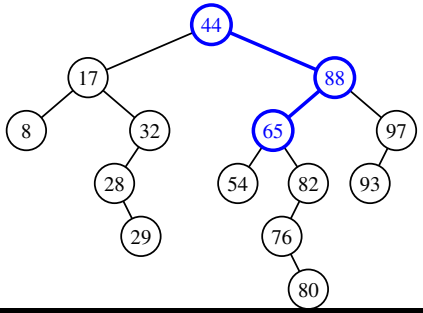
- binarno stablo pretrage je binarno stablo koje čuva (k, v) parove u čvorovima p tako da važi:
  - ključevi koji se nalaze u **levom** podstablu od p su **manji** od k
  - ključevi koji se nalaze u **desnom** podstablu od p su **veći** od k
- listovi ne čuvaju elemente, reference na listove mogu biti None
- inorder obilazak: ključevi u rastućem redosledu



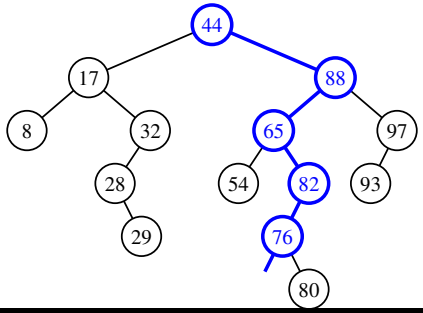
Pretraga u binarnom stablu

- tražimo ključ k polazeći od korena
- idemo levo ako je k manji od tekućeg čvora
- idemo desno ako je k veći od tekućeg čvora
- ako dođemo do lista, k nije nađen

Tražimo 65



Tražimo 68



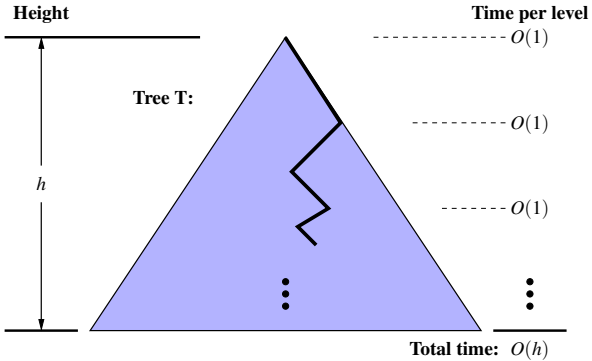
Pretraga u binarnom stablu

TreeSearch(T, p, k)

if $k = p.key$ **then**
  return p {pronađen}
else if $k < p.key \wedge T.left(p) \neq None$ **then**
  return TreeSearch$(T, T.left(p), k)$ {levo podstablo}
else if $k > p.key \wedge T.right(p) \neq None$ **then**
  return TreeSearch$(T, T.right(p), k)$ {desno podstablo}
return None {nije pronađen}

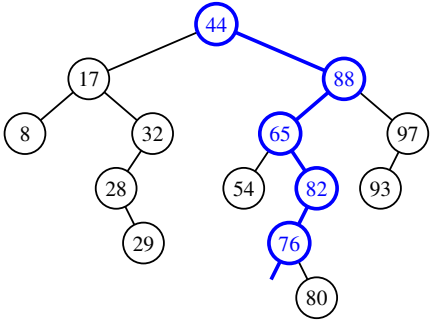
Performanse pretrage u binarnom stablu

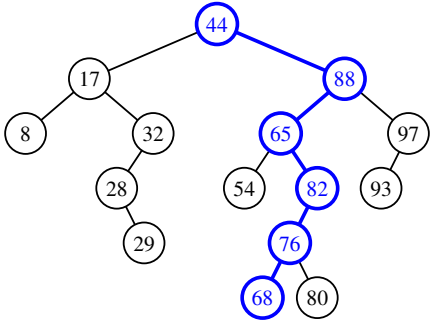
- u svakom rekurzivnom pozivu spuštamo se za jedan nivo u stablu
- testiranje u okviru jednog nivoa je $O(1)$
- ukupan broj testova je $O(h)$, gde je h visina stabla



Dodavanje u stablo

- dodajemo element (k, v)
- prvo tražimo k
- ako k nije u stablu, došli smo do lista gde treba dodati čvor
- primer: dodajemo 68





Dodavanje u stablo

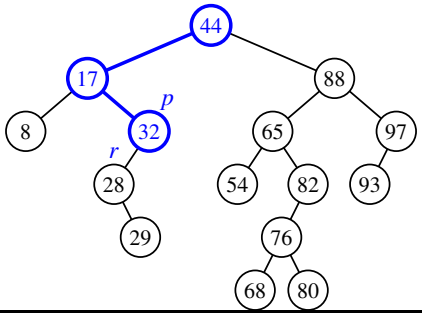
TreeInsert(T, k, v)

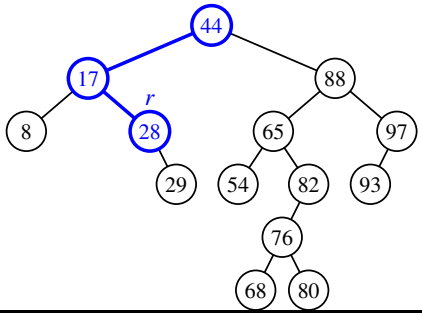
$p \leftarrow$ TreeSearch$(T, T.root, k)$
if $k = p.key$ **then**
  $p.value \leftarrow v$ {ako već postoji zameni vrednost}
else if $k < p.key$ **then**
  p.add_left(k, v) {dodaj levo dete}
else
  p.add_right(k, v) {dodaj desno dete}

- dodaje se uvek u list

Uklanjanje iz stabla

- uklanjamo element sa ključem k
- prvo nađemo p koji sadrži k
- ako p ima **najviše jedno** dete
- njegovo dete r vežemo u stablo umesto njega
- primer: uklanjamo 32



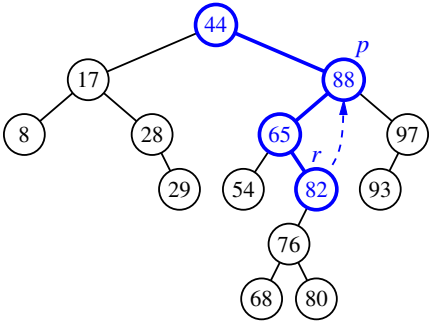


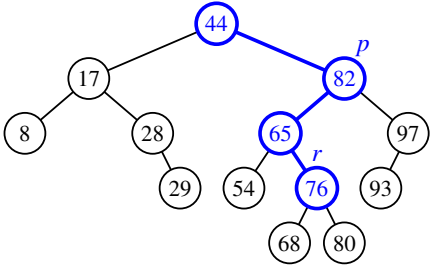
Uklanjanje iz stabla

- ako p ima **dva** deteta
  - nađemo čvor r čiji ključ neposredno prethodi p – to je „najdesniji“ čvor u njegovom levom podstablu
  - vežemo r na mesto p; pošto r neposredno prethodi p po vrednosti ključa, svi elementi u desnom podstablu od p su veći od r i svi elementi u levom podstablu od p su manji od r
  - treba još obrisati stari r – pošto je to „najdesniji“ element, on nema desno dete, pa se može obrisati po prethodnom algoritmu

Uklanjanje iz stabla

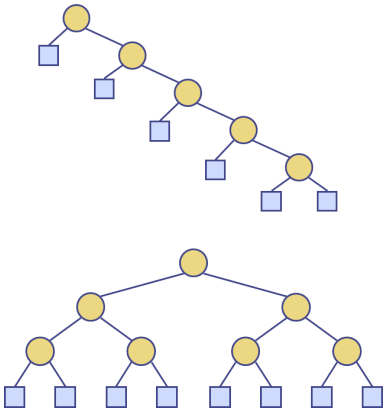
- ako p ima **dva** deteta
- primer: uklanjamo 88





Performanse binarnog stabla pretrage

- zauzeće memorije je $O(n)$
- pretraga, dodavanje i uklanjanje su $O(h)$
- visina stabla h je $O(\log n) \leq h \leq O(n)$



- balansirano stablo ima bolje performanse