Algoritmi in podatkovne strukture – 2

Slovar

Rdeče črna drevesa

Dvojiška (AVL) in B-drevesa

- V obeh primerih mora struktura imeti:
  1. *urejenost*, kar omogoča iskanje (in ostale operacije) po načelu ≫deli in vladaj≪ (lokalnost) in s tem pri iskanju ne zahteva iskanja po celi strukturi; in
  2. *uravnoteženost*, kar omejuje najslabši čas operacij
- pri 1 podobno obe vrsti dreves,
- pri 2 sta različni rešitvi:
  - pri dvojiških drevesih dodajamo vedno nova vozlišča pri listih ter nato uporabimo vrtenja, da drevo uravnotežimo;
  - pri B-drevesih pa dodajamo nova vozlišča pri korenu, kar ohranja liste na isti globini in drevo uravnoteženo.
- prva rešitev (programsko) izgleda precej bolj zapletena kot druga
- ali jo lahko uporabimo pri dvojiških drevesih?

2-3-4 (TTF) drevesa

- vzemimo B-drevesa, ki imajo $b = 4$, se pravi, da imajo v vozlišču do 3 elemente, in elemente v vozliščih

  Pozor, gre torej za prava B drevesa in ne za B+ drevesa.

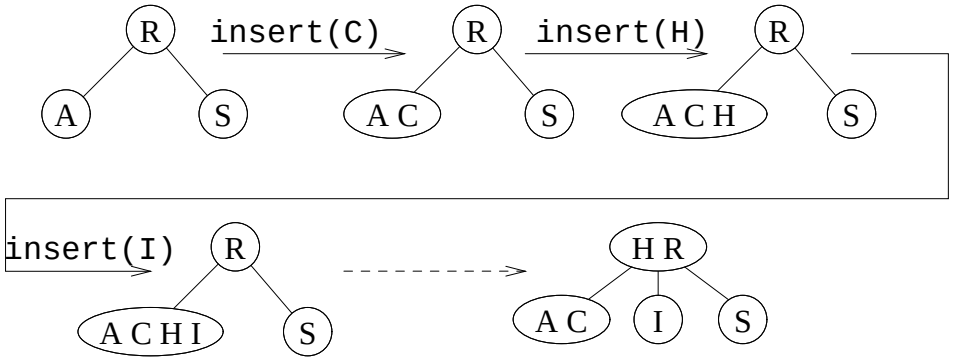
- prava B-drevesa bi sedaj lahko imela v vozlišču 2 ali 3 elemente, mi pa omilimo pogoj in dovolimo tudi 1 element
- dobimo *2-3-4 drevesa* (TTF drevesa) – 2, 3 ali 4 naslednike
- v drevesu višine h je *najmanj* 2^h elementov in če obrnemo, je drevo, ki ima n elementov visoko največ $\lg n$

Operacije nad TTF drevesi

iskanje enako kot pri B-drevesih, ker so drevesa uravnožena in urejena – časovna zahtevnost $O(\log n)$ *primerjav*

vstavljanje pričnemo z (neuspešnim) iskanjem in pridemo do lista nato imamo dve možnosti:

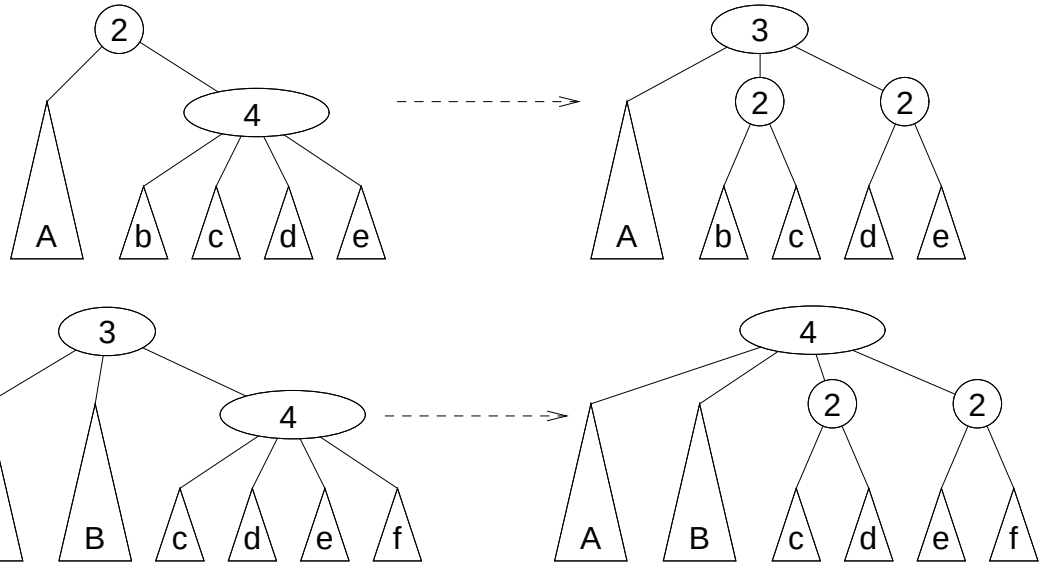
- prišli smo do 2 ali 3 lista: preprosto vstavimo novi element
- prišli smo do 4 lista, ki ga razcepimo in popravljamo vozlišča do korena.



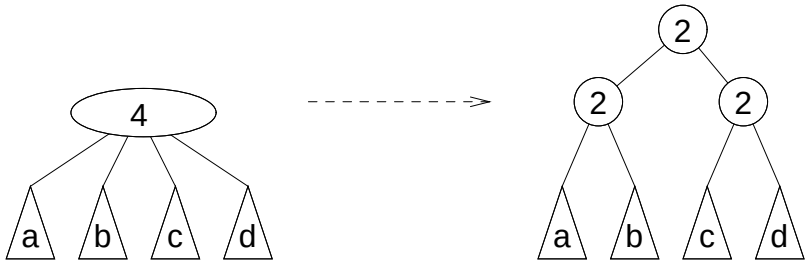
V obeh primerih je časovna zahtevnost $O(\log n)$ *primerjav*.

Lahko pa se drugi možnosti izognemo tako, da zagotovimo, da list nikoli ne bo 4-vozlišče:

na poti navzdol, če srečamo kombinacijo 2 oziroma 3 vozlišča ter 4 vozlišča jo takoj preoblikujemo.



Oziroma, če je koren 4-vozlišče, ga preoblikujemo v 3 2-vozlišča



na ta način nikoli ne dobimo na dnu 4-vozlišča ter nam ni potrebno cepiti staršev.

izločanje tudi enako kot pri B-drevesih – časovna zahtevnost $O(\log n)$ *primerjav*

Primer

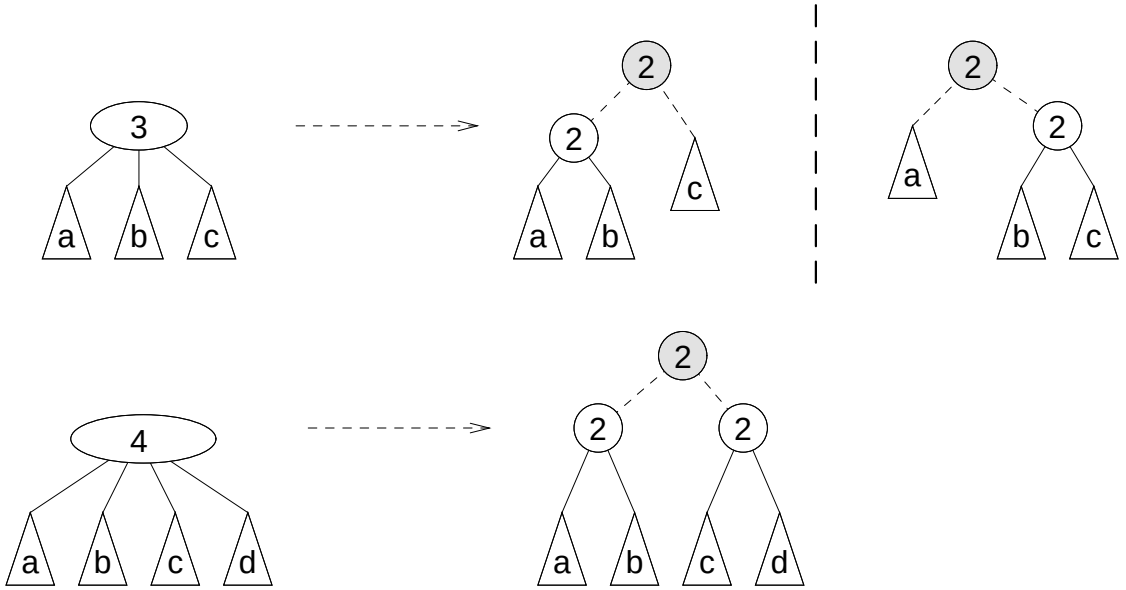
Tvorimo TTF-drevo z zaporednim izboljšanim vstavljanjem ključev:

A S E A R C H I N G

Rdeče črna drevesa – *red-black trees*

- B-drevesa so zelo koristna, če štejemo število dostopov, če pa štejemo število primerjav, se ne obnašajo nič bolje od dvojiških dreves
- zato preoblikujmo naša TTF drevesa v običajna dvojiška drevesa. Posamezna vozlišča preoblikujemo (preslikamo) v dvojiška vozlišča:

  - 2-vozlišča se preprosto preslikajo v dvojiška vozlišča
  - 3-vozlišča se preslikajo v eno od obeh možnosti (a konsistentno uporabljajmo eno, vseeno katero)
  - 4-vozlišča se preslikajo v tri 2-vozlišča
- novo tvorjene povezave označimo za rdeče, stare pa za črne, oziroma tako pobarvamo koren poddrevesa, ki je pod povezavo
- tako dobimo *rdeče-črna poddrevesa*



Lastnosti

(Spodaj so navedene trditve, razmislite o njihovih dokazih!)

- vsi listi (neobstoječa vozlišca – `null`) so črni
- če je vozlišče rdeče, potem sta oba naslednika črna
- vzemimo poljubno (notranje) vozlišče v in vse liste l_i v poddrevesu, kateremu je v koren; potem je število črnih vozlišč na poti od v do lista l_i enako za vse liste poddrevesa
- višina drevesa je največ 2-kratna višina FFT drevesa in zato je še vedno $O(\log n)$
- posledično vse operacije imajo zahtevnost $O(\log n)$ *primerjav* (in *dostopov*)

Zapletenost

	`Find`	`Insert`	`Delete`
seznam	$O(n)$	$O(1)$	$O(n)$
urejen seznam	$O(n)$	$O(n)$	$O(n)$
binarno drevo	$O(n)$	$O(n)$	$O(n)$
AVL drevo	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$
B drevo	$O(\log_b n)$	$O(\log_b n)$	$O(\log_b n)$
RB-drevo	$O(\lg n)$	$O(\lg n)$	$O(\lg n)$

- In koliko je primerjav? Kako velika je konstanta skrita v $O()$?

Primer

Tvorimo RB-drevo z zaporednim vstavljanjem ključev:

A S E A R C H I N G