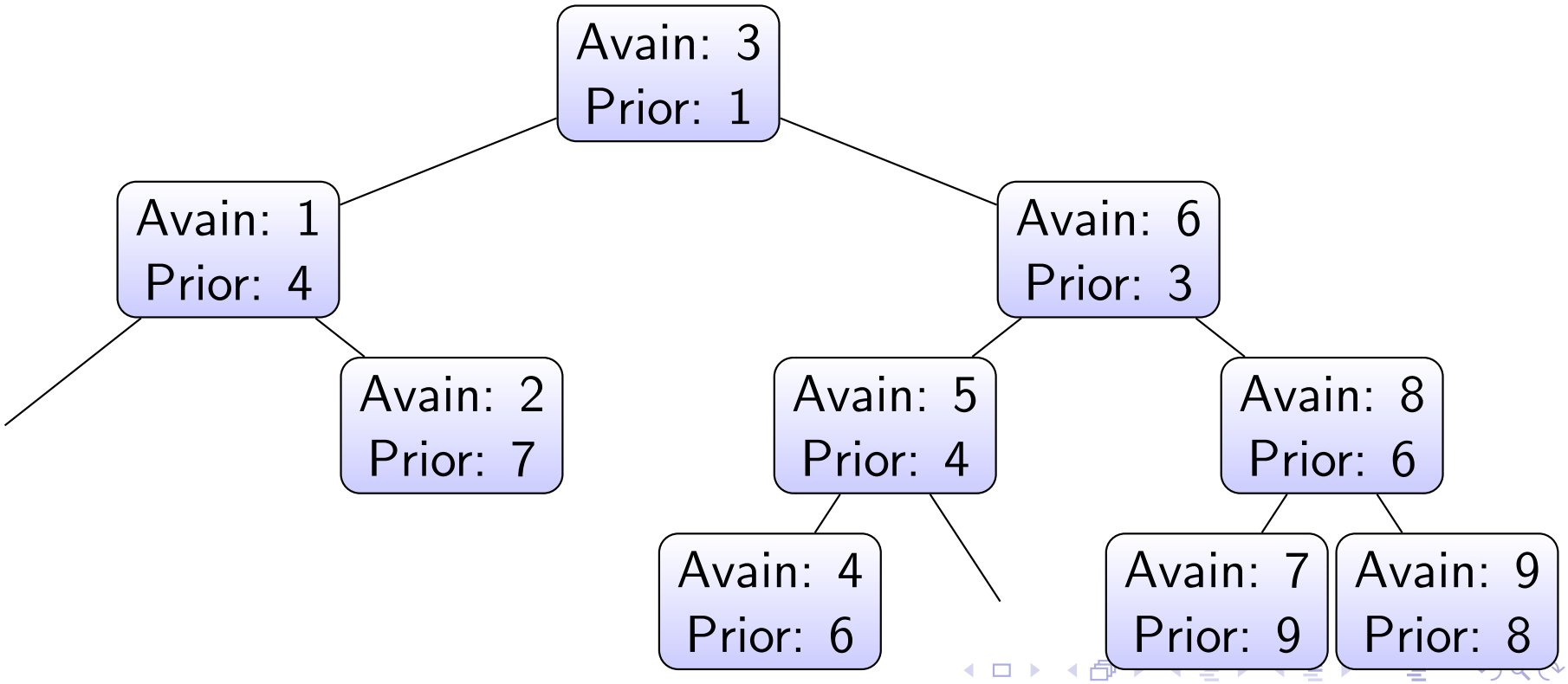
Treap

July 27, 2016

Treap

Satunnaisluvuilla tasapainotettu binäärihakupuu

- Treapin jokaisella elementillä on satunnaisesti määrätty prioriteetti
- Treap on avaimien mukaan binäärihakupuu, eli vasemmassa alipuussa on pienemmät avaimet ja oikeassa isommat
- Treap on prioriteettien mukaan keko, eli solmun lapsien prioriteetit ovat isompia kuin solmun prioriteetit



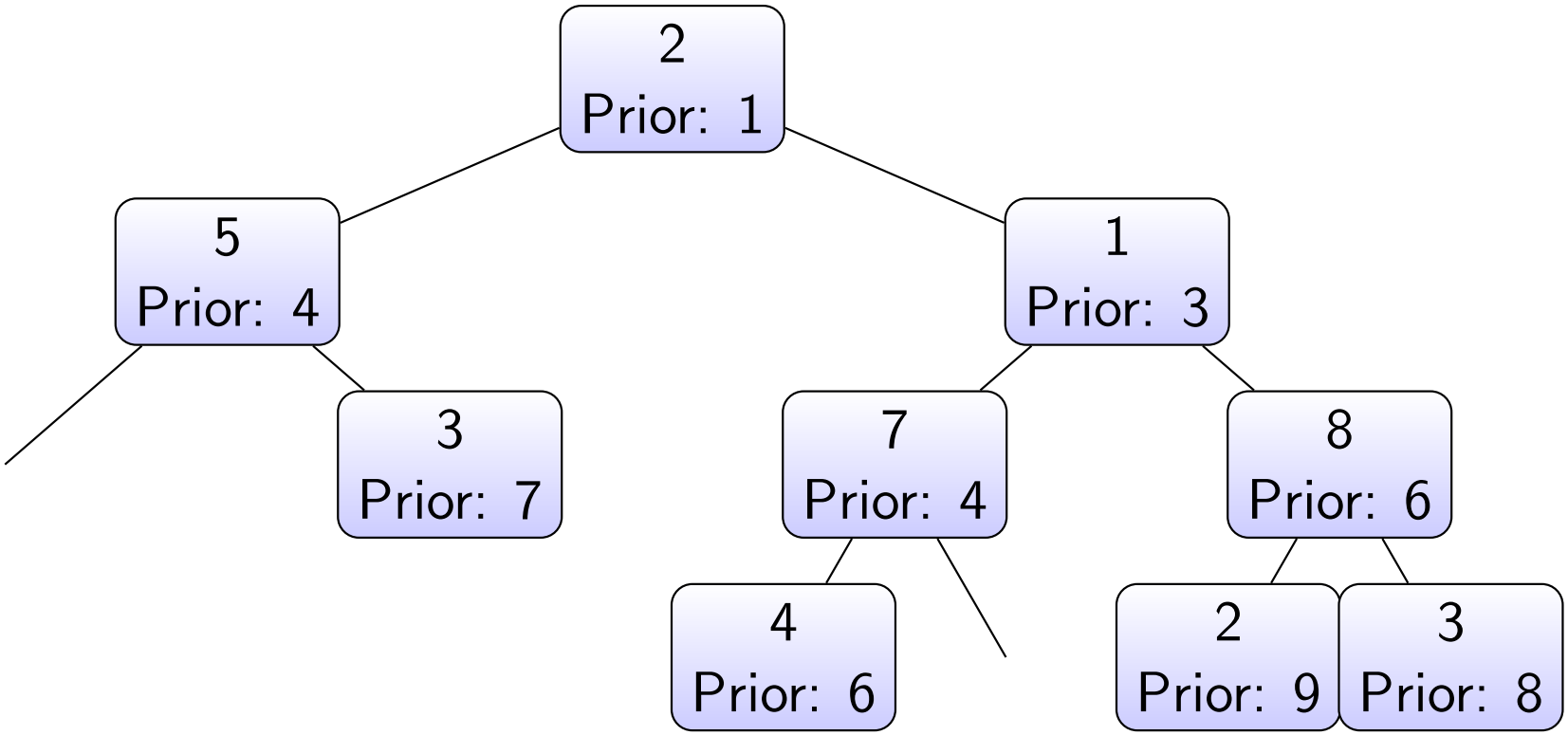
Treap

- Jos kaikki avaimet ja prioriteetit ovat uniikkeja ne määräävät puun yksikäsitteisesti
- Kun prioriteetit ovat satunnaiset, puun korkeuden odotusarvo on $2 \log n$ jossa n on puussa olevien elementtien maara
- On hyvin epätodennäköista että korkeus on moninkertaisesti suurempi kuin $2 \log n$
- Voidaan siis ajatella etta puun korkeus on $O(\log n)$

Käytto

- Kisakoodauksessa tavallista binäärihakupuuta ei tarvitse toteuttaa itse
- Treappia käytetään esittämään dynaamista taulukkoa jota voi leikata ja liimata ja josta voi tehda välikyselyitä
- Ajatellaan että avaimet ovat aina peräkkaiset kokonaisluvut $1..n$ ja solmu jonka avain on i esittää taulukon indeksiä i. Avaimia ei tallenneta puuhun vaan ne selviävät puun sisäjärjestyksestä. Siis sisäjärjestyksessä kohdalla i oleva solmu vastaa taulukon indeksiä i. Sen sijaan taulukon elementit tallennetaan solmuihin. Taulukon elementit eivät vaikuta mitenkään puun rakenteeseen.
- Treapin voi halkaista kahteen puuhun niin että ensimmäiseen puuhun tulee taulukon indeksejä $1..k$ esittävät solmut ja toiseen puuhun taulukon indeksejä $k + 1..n$ esittävät solmut. Halkaiseminen toimii $O(\log n)$ ajassa.

Taulukko treapissa

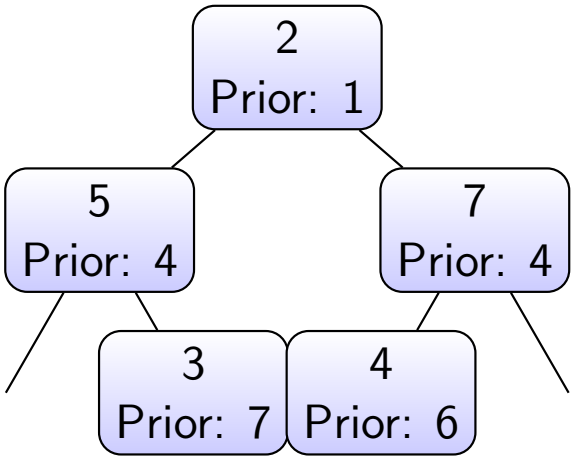


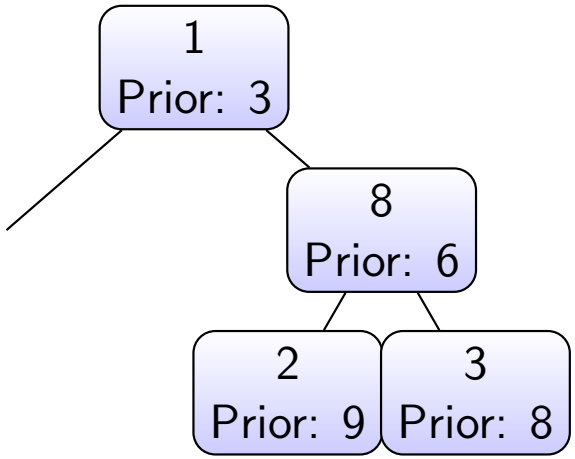
Vastaa taulukkoa

5	3	2	4	7	1	2	8	3

Halkaisu

Nyt halkaistaan taulukko kahteen taulukkoon viidennen elementin kohdalta.





Vastaa taulukkoja

5	3	2	4	7

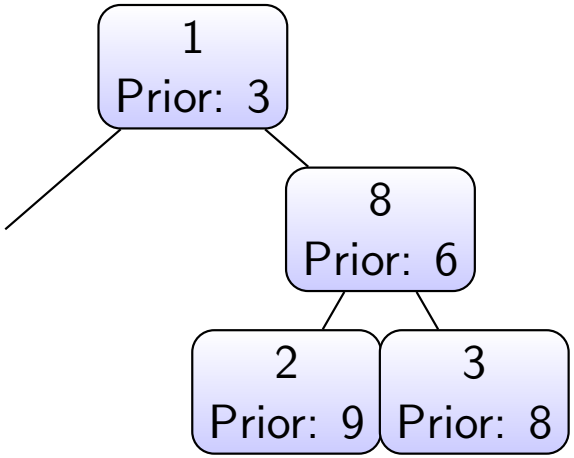
1	2	8	3

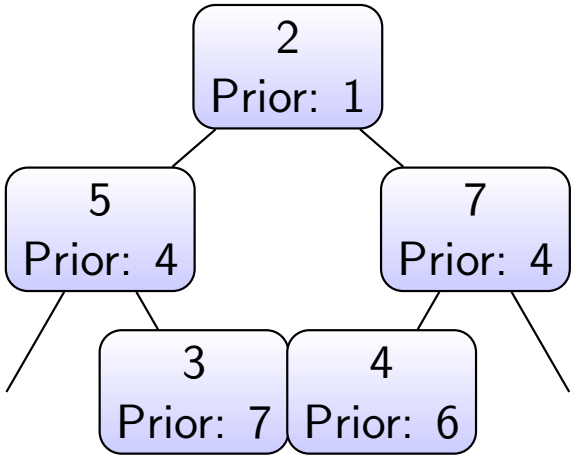
Halkaisu

- Halkaisussa tarvii muuttaa vain solmuja jotka ovat polulla juuresta elementtiin k
- Siis halkaisu tapahtuu $O(\log n)$ ajassa koska puun korkeus on $O(\log n)$
- Prioriteettiehto säilyy halkaisussa, eli halkaisun jälkeenkin solmun vanhemman prioriteetti on pienempi kuin solmun prioriteetti.
- Siis molemmat syntyvät puut ovat $O(\log n)$ korkeita.

Yhdistys

Nyt voidaan vaikka vaihtaa taulukoiden paikkoja ja yhdistaa ne

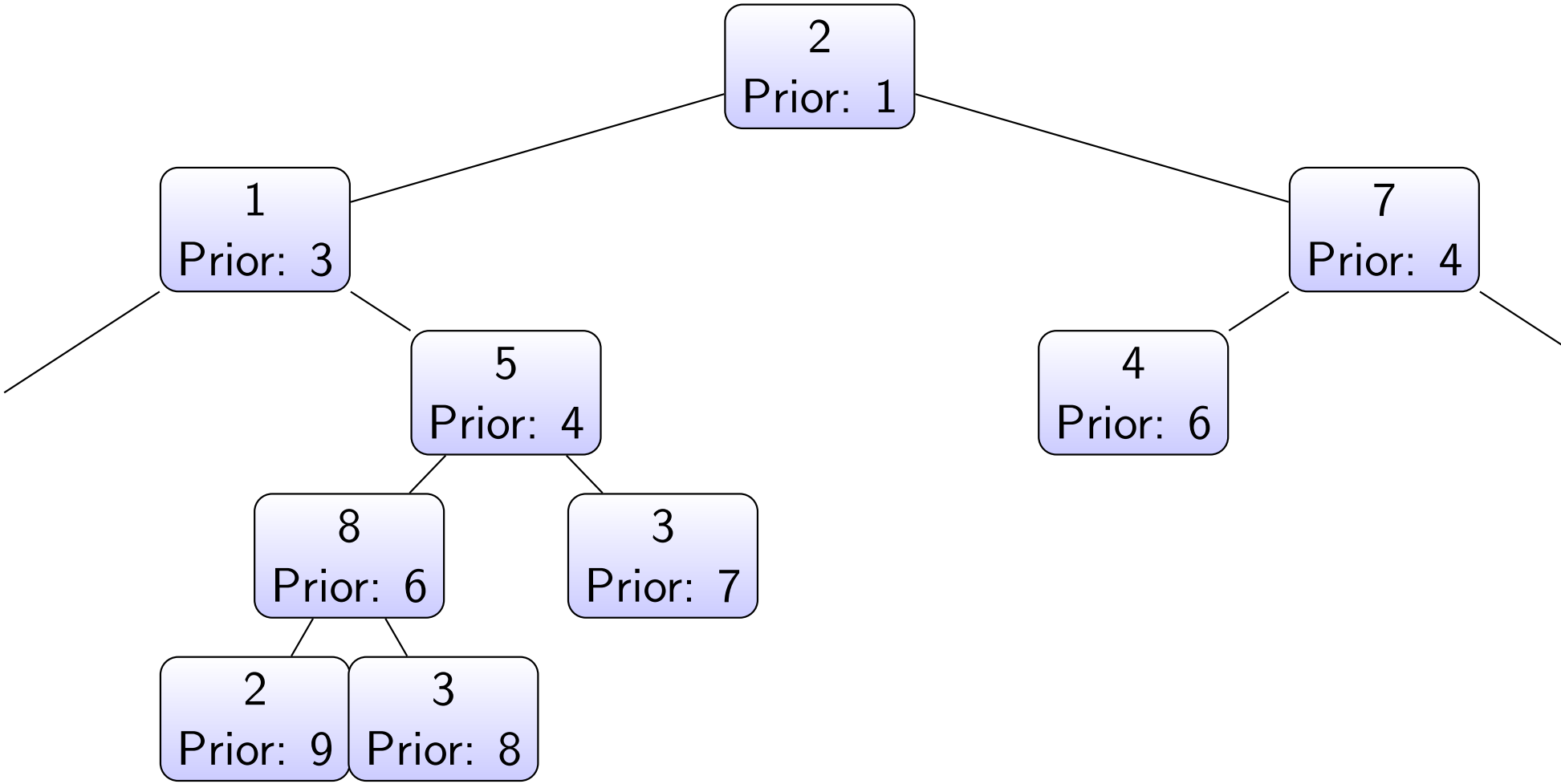




1	2	8	3

5	3	2	4	7

Yhdistys



1	2	8	3	5	3	2	4	7

Yhdistys

- Yhdistys on vaikeampaa sillä pitää huolehtia että kaikki vasemman taulukon elementit ovat oikean taulukon elementtien vasemmalla puolella ja että elementtien järjestys säilyy ja että prioriteettiehto säilyy.
- Tämä onnistuu $O(\log n)$ ajassa matkaamalla yhtä aikaa alaspäin vasemman puun oikeaa reunaa ja oikean puun vasenta reunaa valitsemalla aina pienemmän prioriteetin ensin.

Sovelluksia

- Jos ylläpidetään jokaisessa solmussa sen alipuun solmujen summaa niin voidaan kysellä taulukon välin $[l, r]$ summaa ottamalla kahdella halkaisulla väli $[l, r]$ erilliseksi puuksi, katsomalla sen juuren summan ja yhdistämällä taulukot taas takaisin.
- Laiskojen päivitysten avulla voidaan tehdä esim. välin kääntö takaperin. Binääripuun esittämän taulukon voi kääntää vaihtamalla jokaisen solmun vasemman ja oikean alipuun keskenään. Tämän voi toteuttaa laiskasti ylläpitämällä solmuissa tietoa pitääkö sen alipuu kääntää.