811312A Tietorakenteet ja algoritmit, 2018 - 2019, Harjoitus 5, Ratkaisu

Harjoituksen aihe ovat hash-taulukot ja binääriset etsintäpuut

Tehtävä 5.1 Tallenna avaimet 10,22,31,4,15,28,17 ja 59 hash-taulukkoon, jonka koko on 11, kun tiivistefunktio on $f(k) = k(\text{mod}11)$ ja

a) törmäykset hoidetaan ketjutuksella;
b) käytetään lineaarista luotausta;
c) $g(k) = 1 + (k(\text{mod}10))$ ja käytetään kaksoishashausta funktioiden *f* ja *g* avulla.

Ratkaisu.

a) Lasketaan 10 mod 11 = 10, 22 mod 11 = 0, 31 mod 11 = 9, 4 mod 11 = 4, 15 mod 11 = 4, 28 mod 11 = 6, 17 mod 11 = 6 ja 59 mod 11 = 4.

Siis lopuksi
T[0] -> 22
T[4] -> 4 -> 15 -> 59
T[6] -> 28 -> 17
T[9] -> 31
T[10] -> 10

Muut taulukon paikat ovat tyhjiä.

b) Kun käytetään lineaarista luotausta, h(k,i) = (f(k)+i)(mod 11). Aluksi siis lasketaan f(k), sitten f(k)+1, jne. Aluksi f(10) = 10, joten T[10]=10. Sitten f(22) = 0 ja T[0] =22. Edelleen f(31) = 9, joten T[9] = 31. Jatketaan: f(4) = 4, ja T[4] = 4. Sitten f(15) = 4, joten nyt lasketaan h(k,1) = 5, siis T[5] = 15. Edelleen f(28) = 6, joten T[6] = 28. Vielä f(17) = 6, joten lasketaan h(k,1) = 7. Siis T[7] = 17. Lopuksi f(59) = 4, ja lasketaan h(k,1)=5, h(k,2)=6,h(k,3)=7 ja h(k,4)=8. Siten T[8] = 59.

Lopussa T=

0	1	2	3	4	5	6	7	8	9	10
22				4	15	28	17	59	31	10

c) Nyt h(k,i) = (f(k) + i*g(k))(mod 11), joten luvut 10,22,31 ja 4 sijoittuvat kuten ennenkin. Sen sijaan h(15,0) = 4, h(15,1) = 4+6=10, eivät sijoitu, h(15,2) = 4+12 = 5(mod 11). Siis T[5] = 15. Lisäksi T[6] = 28. Nyt h(17,0) = 6 ja h(17,1) = 6+8 = 3 (mod 11). Lopuksi h(59,0) = 4, h(59,1) = 4+10 = 3 (mod 11), h(59,2) = 4+20= 2 (mod 11) ja T[2] = 59.

Lopussa T=

0	1	2	3	4	5	6	7	8	9	10
22		59	17	4	15	28			31	10

Tehtävä 5.2 Oletetaan, että binääriseen etsintäpuuhun on tallennettu kokonaislukuja väliltä 1..1000. Puusta etsitään lukua 363. Mitkä seuraavista hakujonoista eivät voi olla luvun 363 etsintäjonoja?
Ohje: Tarkastele jokaisen solmun jälkeen lukuväliä, josta etsintää jatketaan.

a) 925, 202, 911, 240, 912, 245, 363.
b) 2, 399, 387, 219, 266, 382, 381, 278, 363.
c) 935, 278, 347, 621, 299, 392, 358, 363.

Ratkaisu. Etsintäjonossa lukuväli, josta etsittävää lukua haetaan, kaventuu koko ajan. Mikäli jonossa esiintyy luku, joka ei ole aiemman välin sisällä, kyseessä ei voi olla oikea etsintäjono.

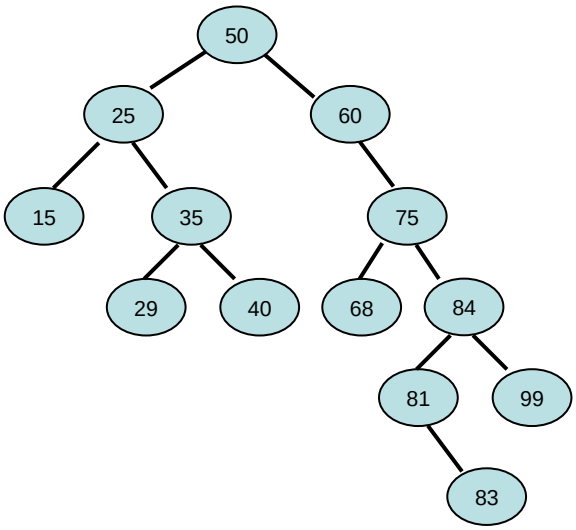
Kohdan a jonolle 925, 202, 911, 240, 912, 245, 363 saadaan välit:
$(-\infty,925]$, [202, 925], [202,911], [240,911], 912 ei kuulu edelliseen väliin, joten kysymyksessä ei ole oikea etsintäjono.

Kohdassa b jono on 2, 399, 387, 219, 266, 382, 381, 278, 363 ja välit ovat:
$[2,\infty)$, [2, 399], [2,387], [219,387], [266,387], [266,382], [266,381], [278,381], 363 löytyi. Jono on mahdollinen etsintäjono.

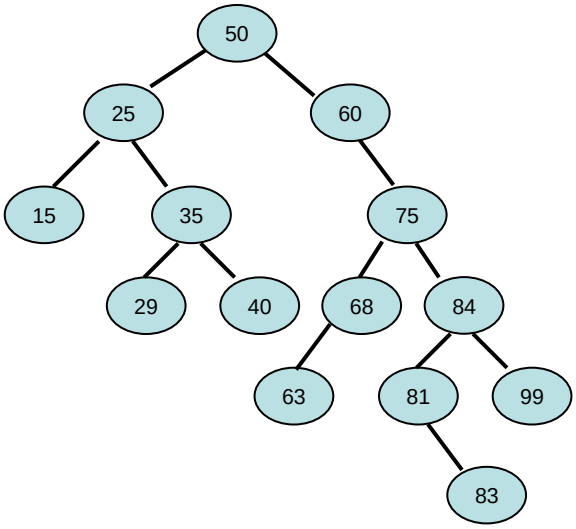
Kohdan c jonossa 935, 278, 347, 621, 299, 392, 358, 363 välit ovat:
$(-\infty,935]$, [278,935], [347,935], [347,621], 299 ei kuulu edelliseen väliin, joten kysymyksessä ei ole oikea etsintäjono.

Siis: jonot a ja c eivät voi olla luvun 363 etsintäjonoja.

Tehtävä 5.3 Lisää ensin alla olevasta binääriseen etsintäpuuhun avain 63 ja poista tämän jälkeen avaimet 29, 35, 75 ja 60 tässä järjestyksessä. Esitä ratkaisut kuvioilla ja perustele niitä algoritmeihin BST_POISTA ja BST_TRANSPLANT nojaten.



Ratkaisu. Avain lisätään aina lehtisolmuksi sopivaan paikkaan puussa, joten avaimen 63 lisäämisen jälkeen puu on

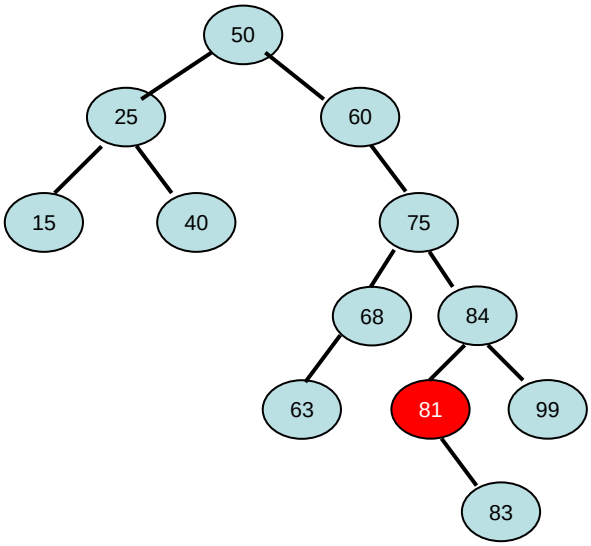


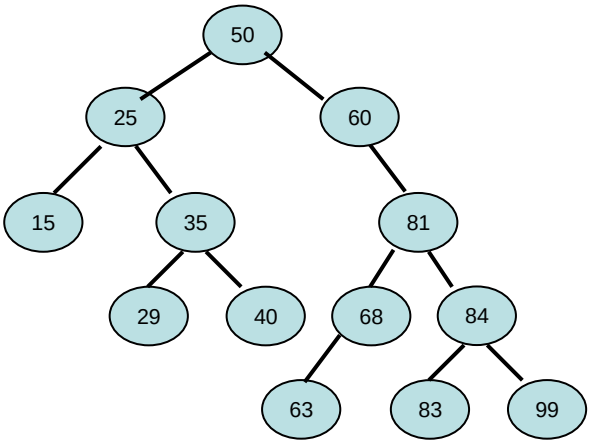
Algoritmi solmun poistamiseksi on seuraava:

```
BST_POISTA(T,k)                          8.    else
1.     x = BST_ETSI(T,k)                 9.      y = BST_MINIMI(x.right)
2.     if x == NIL                       10.     if y.p != x
3.       return FALSE                    11.       BST_TRANSPLANT(T,y,y.right)
4.     if x.left == NIL                  12.       y.right = x.right
5.       BST_TRANSPLANT(T,x,x.right)     13.       y.right.p = y
6.     else if x.right == NIL            14.     BST_TRANSPLANT(T,x,y)
7.       BST_TRANSPLANT(T,x,x.left)      15.     y.left = x.left
                                         16.     y.left.p = y
                                         17.   return TRUE
```

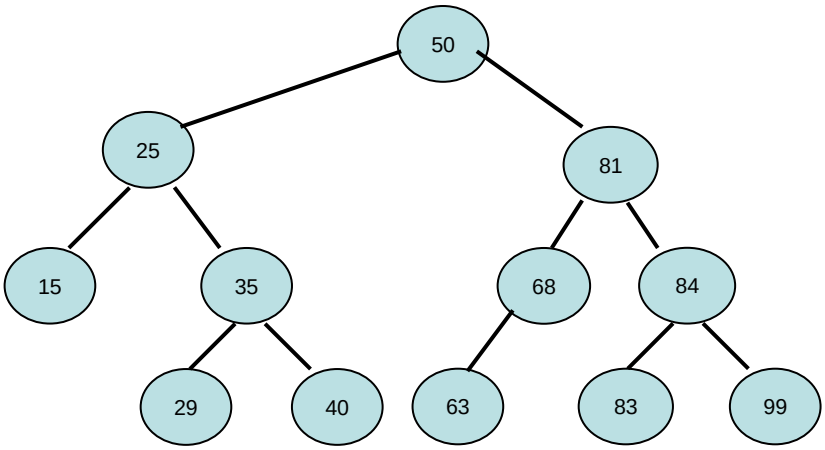
Algoritmi BST_TRANSPLANT(T,u,v) siirtää solmusta v lähtevän alipuun solmusta u lähtevän alipuun tilalle ja BST_MINIMI(u) hakee solmusta u lähtevän alipuun pieniarvoisimman solmun.

Avain 29 voidaan poistaa puun lehdestä suoraan (algoritmin rivi 5). Tämän jälkeen avaimen 35 solmulle jää ainoastaan yksi alipuu, joten avain poistetaan siirtämällä sen äitisolmun linkki osoittamaan poistettavan solmun lapsisolmuun (samoin algoritmin rivi 5). Tuloksena on alla olevan kuvan vasemmanpuoleinen puu. Tähän puuhun on merkitty seuraavaksi poistettavan avaimen 75 seuraaja 81 (haetaan algoritmin rivillä 9). Ensin poistetaan seuraajasolmu, jolla ei ole vasenta alipuuta. Näin ollen se voidaan poistaa siirtämällä sen äitisolmun vasen osoitin seuraajasolmun oikeaan alipuuhun (algoritmin rivit 11-13). Lopuksi kopioidaan seuraajan avain poistettavan avaimen 75 paikalle (algoritmin rivit 14-16). Tuloksena on oikeanpuoleinen puu.





Avain 60 on jälleen solmussa, jolla on vain yksi alipuu, joten se voidaan poistaa äitisolmun linkkiä siirtämällä (algoritmin rivi 5) ja saadaan alla oleva puu



Tehtävä 5.4 Alle linkitetyssä tekstitiedostossa ” task5_4_nums.txt” on tallennettuna 25000 satunnaislukua väliltä [1,1000000]. Jokainen luku on tallennettu omalle rivilleen. Tehtävänä on lukea luvut sellaiseen tallennusrakenteeseen, että annetusta luvusta voidaan nopeasti todeta, onko se tiedostossa olevien lukujen joukossa. Testaa ohjelmaasi luvuilla 613695, 906429, 180551, 151841, 951585 ja 569127, joista kolme ensimmäistä eivät ole tiedostossa ja kolme viimeistä ovat. Valitse toinen alla olevista vaihtoehdoista:

a) C-kielisessä ohjelmassa käytetään tallennusrakenteena hash-taulukkoa, jonka koko on 49999. Hash-funktio saadaan käyttämällä jakomenetelmää ja törmäysten hallintaan käytetään lineaarista luotausta. Alle linkitetyssä tiedostossa ”task5_4_base.c” on ohjelma, jota voi käyttää toteutuksessa pohjana. Jos aikaa vielä jää, niin voit testata vielä ohjelmasi suorituskykyä tuottamalla 100 000 satunnaislukua väliltä [1,1000 000] ja mittaamalla näiden hakemiseen kulunut aika.
b) Python-kielisessä ohjelmassa käytetään tallennusrakenteena binääristä hakupuuta, jonka toteutus on tiedostossa ”bst.py”. Funktio bst_insert lisää puuhun avaimen ja bst_search etsii avainta. Jos avainta ei löydy, paluuarvo on None. Voit aloittaa ohjelman laatimisen tiedostoon ” task5_4_base.py”. Testaa vielä ohjelmasi suorituskykyä tuottamalla 100 000 satunnaislukua väliltä [1,1000 000] ja mittaamalla näiden hakemiseen kulunut aika.

Ratkaisu. Alla olevissa tiedostoissa on toteutettu vaadittavat ohjelmat. C-kielisen ohjelman funktio insert_number palauttaa arvon 1, jos tallentaminen onnistuu (tai jos luku on jo tallennettu). Mikäli taulukko on täynnä, palautetaan arvo -1. Lukua taulukosta etsivä funktio search_number palauttaa luvun paikan taulukossa, mikäli luku on tallennettuna siihen. Jos luku ei ole taulukossa, niin palautetaan arvo -1. Huomaa, miten funktioissa on otettu huomioon lineaarisen luotauksen vaikutus. Kun testikoneella mitattiin 100 000 luvun hakemiseen kulunutta aikaa, saatiin tulokseksi 0.0090 sekuntia. Aikaa 100 000 000 luvun hakemiseen kului 8.18 sekuntia.

Python-ohjelmassa on hyödynnetty valmista binäärisen hakupuun toteutusta. Python-ohjelmalta kului testikoneella 100 000 luvun etsimiseen puusta 0.71 sekuntia ja 1000 000 luvun etsimiseen meni 7.37 sekuntia.