Hakupuut

- tässä luvussa tarkastelemme puita tiedon tallennusrakenteina
- hakupuun avulla voidaan toteuttaa kaikki $joukko$-tietotyypin operaatiot (myös **succ** ja **pred**)
- pahimman tapauksen aikavaativuus on tavallisella hakupuulla $O(n)$ kaikilla operaatioilla
- käyttämällä tasapainotusta saadaan kaikki operaatiot aikaan $O(\log n)$
- tämä on eräs keskeisimpiä tuloksia tietorakenteiden teoriassa, ja menetelmät ovat tärkeitä myös sovelluksissa
- hakupuun perusversio on helppo toteuttaa, tasapainotus hivenen monimutkaista

Puu on käsitteenä tärkeä muutenkin kuin *joukko*-toteutuksena

- verkkojen (eli graafien) teoriassa puu tarkoittaa yleensä yhtenäistä syklitöntä verkkoa
  - verkkoteoriassa tässä luvussa käsiteltävä puu olisi lähinnä *juurellinen järjestetty puu*
- puu on hyvä malli erilaisille hierarkioille
  - dokumentti, sen luvut, aliluvut jne.
  - organisaatiokaavio
  - ...
- algoritmiikassa se on hyvä malli haarautuvan suorituksen logiikalle

Peruskäsitteitä

- solmu, kaari (vastaavat käsitteet kuin verkkoteoriassa)
- juuri, alipuu, alipuun juuri
- vanhempi, lapsi, edeltäjä, jälkeläinen, . . .
- sisäsolmu, lehti
- polun pituus on sen kaarten lukumäärä
- solmusta itseensä on aina nollan pituinen polku
- solmun syvyys eli taso on siihen juuresta johtavan polun pituus
- siis juuren syvyys on nolla
- solmun korkeus on pisimmän polun pituus solmusta sen alipuussa sijaitsevaan lehteen
- siis lehden korkeus on nolla
- puun korkeus on sen juuren korkeus

Puun rekursiivinen määritelmä

- On olemassa tyhjä puu (merkitään yleensä NIL)
- Jos r on mikä tahansa solmu ja $T_1, \ldots, T_k$ ovat puita, niin $\langle r, T_1, \ldots, T_k \rangle$ on puu, jonka juuri on r ja alipuut $T_1, \ldots, T_k$.

Tämä rekursiisivinen formaali määritelmä ei ehkä ole kovin intuitiivinen, mutta antaa rungon monille rekursiivisille algoritmeille

Huomaa, että alipuiden järjestyksellä on väliä.

Binääripuu

- binääripuussa solmulla on korkeintaan kaksi lasta,
- tarkemmin, solmulla voi olla vasen ja oikea lapsi, joista toinen tai molemmat voi puuttua
- yksilapsisen solmun tapauksessa *on* eri asia, onko lapsi vasen vai oikea
- käytännön toteutuksessa on yleensä järkevää ajatella, että jokaisella solmulla on aina tasan kaksi alipuuta, joista toinen tai molemmat voi olla NIL
- solmun x vasemman alipuun juureen osoittaa $x.\mathit{left}$ ja oikean alipuun juureen $x.\mathit{right}$
- jos alipuu puuttuu, vastaava osoitin on NIL

Binäärihakupuu

- solmussa x on avain $x.key$, joka on kokonaisluku tai muun järjestetyn joukon alkio (esim. merkkijono)
- lisäksi solmuun voi liittyä muuta dataa, jonka taas jätämme merkitsemättä
- solmusta $x.left$ alkavassa alipuussa kaikki avaimet ovat korkeintaan $x.key$
- solmusta $x.right$ alkavassa alipuussa kaikki avaimet ovat vähintään $x.key$
- usein on kätevää liittää solmuun myös $x.parent$
- puun T juureen osoittaa $T.root$
- puuttuvaa lasta vastaava osoitin on NIL, samoin juuren $parent$

Binääripuun läpikäynti

- sisäjärjestys (inorder)
  1. käsittele vasen alipuu
  2. käsittele juuri
  3. käsittele oikea alipuu

- esijärjestys (preorder)
  1. käsittele juuri
  2. käsittele vasen alipuu
  3. käsittele oikea alipuu

- jälkijärjestys (postorder)
  1. käsittele vasen alipuu
  2. käsittele oikea alipuu
  3. käsittele juuri

Esimerkki Solmujen tulostaminen sisäjärjestyksessä

tulosta-alkiot(x)
    if $x \neq$ NIL
        tulosta-alkiot($x.\,left$)
        print $x.\,key$
        tulosta-alkiot($x.\,right$)

Erityisesti jos puu T on hakupuu, niin tulosta-alkiot($T.\,root$) tulostaa sen solmut järjestyksessä pienimmästä suurimpaan.

Esimerkki Binääripuun solmujen laskeminen

lukumäärä(x)
    if $x =$ NIL
        return 0
    else
        return lukumäärä($x.\,left$) + lukumäärä($x.\,right$) + 1

Lause Binääripuun tasolla i on korkeintaan 2^i solmua.

Lause Jos binääripuun korkeus on h, siinä on enintään $2^{h+1} - 1$ solmua.

Lause Jos binääripuun solmujen lukumäärä on n, sen korkeus on enintään $n - 1$.

Lause Jos binääripuun solmujen lukumäärä on n, sen korkeus on vähintään $\log_2(n + 1) - 1$.

- täysi (full) binääripuu: jokaisella solmulla nolla tai kaksi lasta (tai ei-tyhjää alipuuta)
- täydellinen (complete) binääripuu: täysi, ja lisäksi kaikki lehdet samalla tasolla
- melkein täydellinen: täydellinen paitsi ehkä alin taso

Joukko-operaatioiden toteuttaminen

search:

- Lähdetään liikkeelle juuresta.
- Edetään puussa alaspäin, kunnes
  - osutaan solmuun, jossa on haluttu avain, ja palautetaan osoitin solmuun; **tai**
  - päädytään tyhjään alipuuhun ja palautetaan NIL
- Alaspäin mennessä valitaan oikea tai vasen haara sen mukaan, onko solmun avain pienempi vai suurempi kuin valittu.

Esimerkin vuoksi esitetään tämä pseudokoodina.

Avain k etsitään puusta T kutsulla $search(T.root, k)$.

Rekursiivinen toteutus:

```
search(x,k)
    if x == NIL or x.key == k
        return x
    if k < x.key
        return search(x.left,k)
    else return search(x.right,k)
```

Rekursio on tässä helppo välttää:

```
search(x,k)
    while x ≠ NIL and x.key ≠ k
        if k < x.key
            x = x.left          // siirretään x viittaamaan vasempaan lapseen
        else x = x.right        // siirretään x viittaamaan oikeaan lapseen
    return x
```

Pahimman tapauksen aikavaativuus on $O(h)$, missä h on puun korkeus.

Tilavaativuus on $O(h)$ rekursiivisessa ja $O(1)$ iteratiivisessa toteutuksessa.

min: edetään puussa alas vasemmalle niin pitkälle kuin pääsee

max: edetään puussa alas oikealle niin pitkälle kuin pääsee

succ: solmua x seuraavan solmun löytämiseksi

- jos solmulla x on ei-tyhjä oikea alipuu, etsitään sen min
- muuten edetään puussa ylöspäin (seuraten $parent$-osoittimia) niin kaun kun vastaantulevat avaimet ovat pienempiä kuin $x.key$
- ensimmäinen y, jolla $y.key$ on suurempi kuin $x.key$, on x:n seuraaja

pred: edellisen peilikuva

insert :

- oletamme, että avain ei ole puussa (tarkista tämä tekemällä search ennen insertiä jos aiheellista)
- edetään alaspäin kuten search
- koska avain ei ole puussa, päädytään tyhjään alipuuhun
- laitetaan tyhjän alipuun paikalle solmu, jossa lisättävä avain
- huom. täytyy pitää muistissa osoitin lisäyskohdan vanhempaan, että osoittimet saadaan paikalleen

delete : monimutkaisin operaatio

- oletetaan annetuksi osoitin x poistettavaan solmuun
- jos x on lehti, solmu voidaan poistaa ilman komplikaatioita
- jos x:llä on tasan yksi lapsi, se voidaan ohittaa puussa tyyliin

  x.left.parent = x.parent
  x.parent.right = x.left

  (pari eri tapausta sen mukaan, mitkä ovat vasempia ja oikeita lapsia)
- jos x:llä on kaksi lasta, tilanne on hieman monimutkaisempi

delete(x) kun x:llä on kaksi lasta:

- idea on säilyttää solmuun x liittyvät linkit osana puurakennetta, mutta vaihtaa avain
- jos y on solmun x seuraaja, niin haluttu lopputulos saadaan asettamalla $x.key = y.key$ ja poistamalla solmu y
  - koska solmujen x ja y avaimet ovat peräkkäiset, niin x:n avaimen korvaaminen y:n avaimella ei riko hakupuun ehtoja
- seuraajasolmulla y on korkeintaan yksi lapsi, joten sen poistaminen palautuu edellä käsiteltyihin helppoihin tapauksiin
  - koska x:llä on kaksi lasta, y on x:n oikeassa alipuussa
  - nyt y:llä ei voi olla vasenta lasta, koska se olisi järjestyksessä x:n ja y:n välissä

- kaikkien operaatioiden aikavaativuus on $O(h)$, missä h on puun korkeus, ja tilavaativuus $O(1)$

- h on parhaimmillaan $O(\log n)$ ja pahimmillaan $O(n)$, missä n on avainten lukumäärä puussa

- voidaan osoittaa (ei kuulu tämän kurssin asioihin), että jos avaimet lisätään satunnaisessa järjestyksessä, niin h on keskimäärin $O(\log n)$

- kuitenkin $h = \Omega(n)$ esim. jos avaimet lisätään suuruusjärjestyksessä

- n-solmuinen hakupuu voidaan aina uudelleenorganisoida niin, että korkeudeksi tulee $O(\log n)$, mutta tämä vie aikaa

- hakupuun tasapainottamisessa puuhun lisätään kirjanpitotietoa, jonka avulla sen rakennetta voidaan insert- ja delete-operaatioiden yhteydessä tarvittaessa ”korjata” niin, että korkeus pysyy $O(\log n)$:ssä

- hakupuiden tasapainotusmenettelyistä esitämme tällä kurssilla AVL-puut; vaihtoehtoja ovat esim. punamustat puut ja B-puut (tärkeitä tietokannoissa)

- siis tasapainotetuilla hakupuilla kaikki joukko-operaatiot toimivat ajassa $O(\log n)$