Otsingumeetodid

Järjestotsing. Kahendotsing. Puuotsing. Otsingupuud.

Otsinguülesanne

On antud N kirjet. Igal kirjel on võtmeväli, mis on reeglina unikaalne.

Kirjete hulgast on vaja leida üks kirje, mille võti vastab otsitavale võtmele K. Otsitav kirje tagastatakse või antakse teada, et kirjet võtmega K ei leitud.

Edukas otsing – leiti kirje võtmeväärtusega K

Mitteedukas otsing – ei leitud kirjet võtmega K.

Otsingumeetodid

Meetodite keerukusklassid varieeruvad lineaarsest kuni konstantseni (O(N) ... O(1)).

Sõltuvalt andmestiku iseloomust, sobib üks või teine meetod.

- Kas andmestik on muutumatu või muutuv?
- Kas andmed on staatiliselt massiivis või dünaamiliselt ahelas või on moodustatud mingi erilise iseloomuga struktuur?
- Kirjete arv?

Otsingu efektiivsuse tõstmiseks kasutatakse ka spetsiaalseid andmestruktuure.

Järjestotsing

Järjestotsing ehk **jadaotsing** *(sequential search)* on lihtne, kuid aeglane otsingumeetod.

Kuidas toimub järjestotsing?

Keerukus: O(N) (mida see tähendas?)

Millal sobib kasutada? Kas on piiranguid kasutamisel?

Kahendotsing

Kahendotsing *(binary search)*

Kuidas toimub kahendotsing?

Milline on meetodi keerukusklass?

Võrreldes järjestotsinguga – millised on eelised ja puudused?

Millised on piirangud kasutamisel?

Kahendotsingupuu

Kahendotsingupuu *(binary seach tree)* on kahendpuu sellise kirjete paigutusega, mis tagab nii andmete otsimisel, lisamisel kui ka kustutamisel keerukuse O(log N).

Sobilik muutuva andmestiku korral.

Tuleta meelde, mis on kahendpuu!

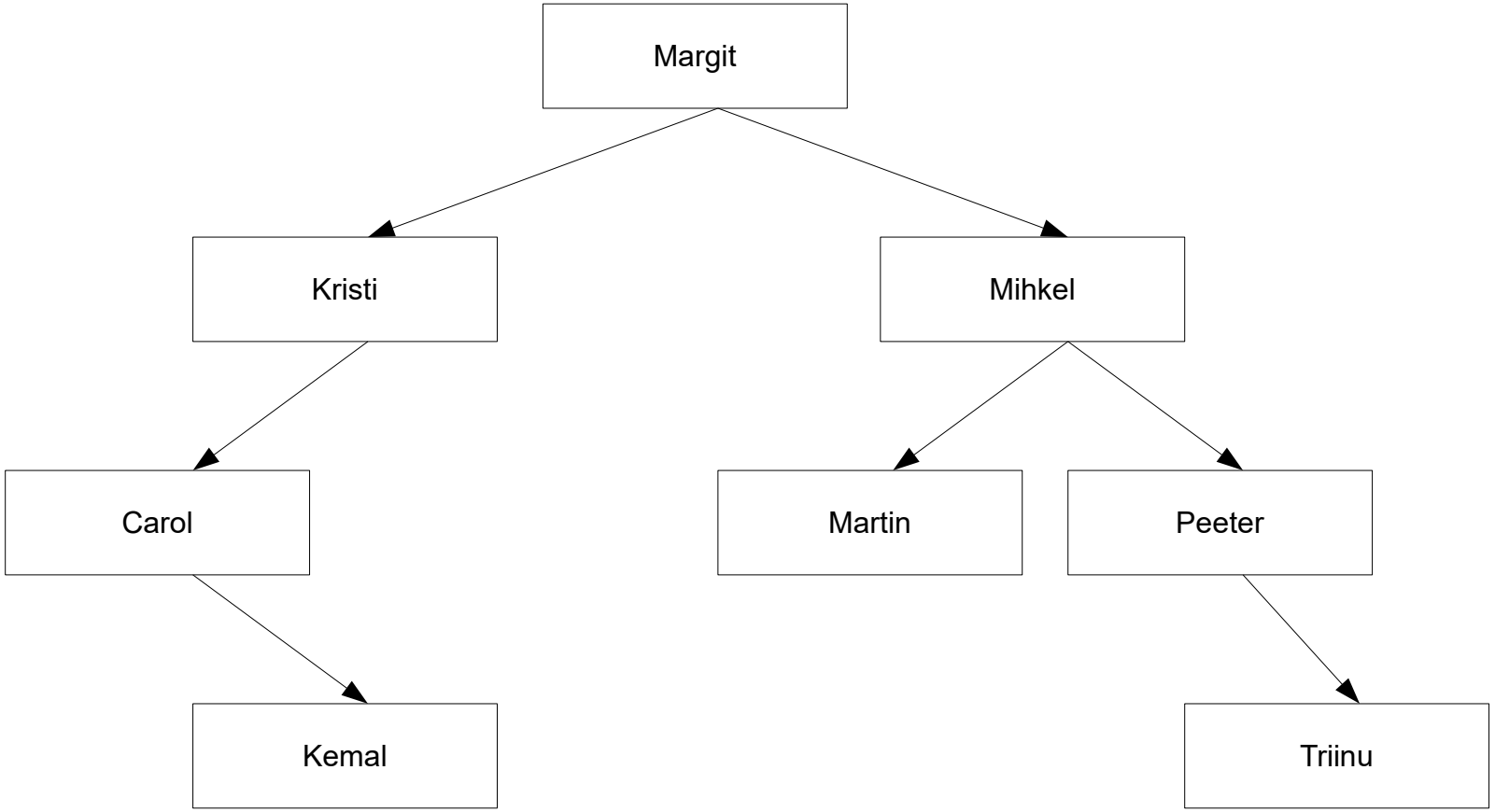
Reeglid võtmete paigutamiseks

Kahendpuu on kahendotsingupuu, kui võtmed on puusse paigutatud järgmiste reeglite järgi:

- iga tipu vasakpoolse lapse võti on väiksem tipu võtmest (kõik tipu vasakpoolses alampuus olevad võtmed on väiksemad tipu võtmest);
- iga tipu parempoolse lapse võti on suurem tipu võtmest (kõik tipu parempoolses alampuus olevad võtmed on suuremad tipu võtmest);

Kahendotsingupuu näide

Võtmed on lisatud selles järjekorras: Margit, Kristi, Mihkel, Peeter, Martin, Carol, Triinu, Kemal



Ülesanne

Lisa eelmisel slaidil olevad nimed puusse teistsuguses järjekorras:

Kristi, Mihkel, Peeter, Carol, Kemal, Triinu, Margit, Martin

Milline puu nüüd tekib?

Kahendotsingupuu operatsioonid

Üldiselt kasutatavad operatsioonid:

- otsimine *(search);*
- vähim ja suurim *(minium, maximum)*;
- lisamine *(insert);*
- kustutamine *(delete);*
- suuruselt eelmine ja järgmine *(predecessor, successor)*;
- sorteerimine (sort).

Kahendotsingupuud sobib kasutada nii sõnastikuks *(dictionary)* ja eelisjärjekorraks *(priority queue)*.

Kahendpuu mälus

Allikas: Cormen et al, "Introduction to algorithms", Third Edition

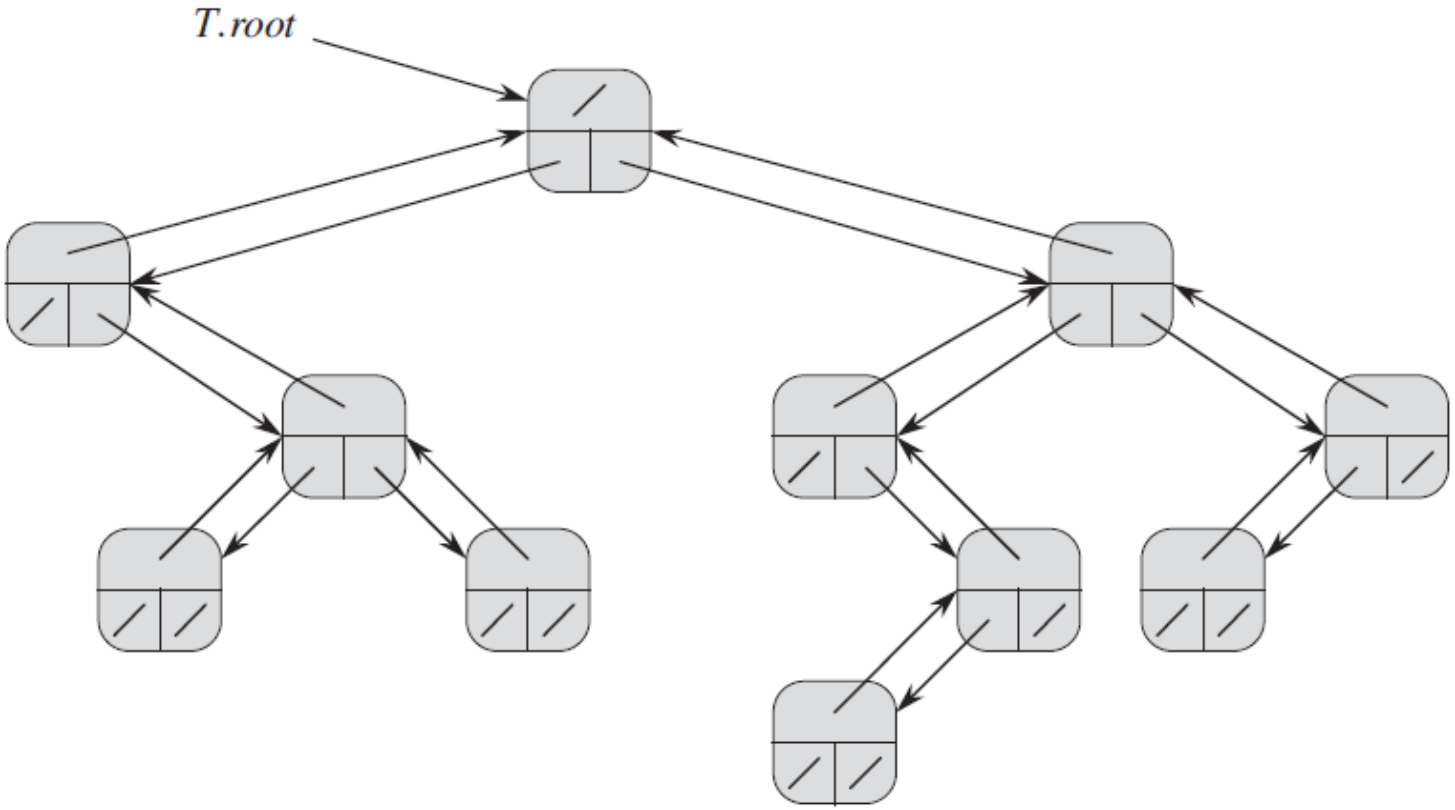


Figure 10.9 The representation of a binary tree T. Each node x has the attributes $x.p$ (top), $x.left$ (lower left), and $x.right$ (lower right). The *key* attributes are not shown.

Otsing

Otsimist alustatakse alati puu juurelemendist ja toimitakse järgmiselt:

- Kui otsinguvõti on väiksem tipus olevast võtmest, liigu vasakusse alampuusse
- Kui otsinguvõti on suurem tipus olevast võtmest, liigu paremasse alampuusse
- Kui võtmed on võrdsed, on kirje leitud
- Kui edasi liikuda ei saa, siis otsitav kirje puudub.

Algoritmile on sisendiks otsinguvõti K ja puu juure aadress T.

Otsingu algoritm 1

See algoritm, nagu ka kõik järgmised, esitatakse pseudokeeles vastavalt raamatule *(Cormen et al, "Introduction to Algorithms", Third Edition).* Algoritm tagastab viida otsitavale elemendile või NIL, kui elementi ei leitud).

```
Tree-Search(k, T)
    x = T
    while x != NIL and k != x.key do
        if k < x.key
            x = x.left
        else
            x = x.right
    return x
```

Otsingu algoritm 2

Otsingu algoritmi rekursiivne variant (tagastab viida otsitavale elemendile või NIL, kui elementi ei leitud)

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

Tipu (võtme) lisamine

Uue võtme lisamine peab arvestama puu reeglitega, st igale võtmele tuleb leida õige koht.

Koha leidmiseks tehakse otsimisega sarnased sammud, kasutades võrdlemiseks lisatava kirje võtit.

Kui alampuusse minna ei saa, sest see on tühi, on uuele võtmele õige koht leitud.

Uus võti lisatakse alati leheks.

Kirje lisamise algoritm

Algoritm eeldab, et vastava võtmega kirjet ei ole olemas. Uue elemendi aadress on z.

```
TREE-INSERT(T, z)
y = NIL
x = T.root
while x != NIL          // liigume puus õigesse kohta
    y = x
    if z.key < x.key
        x = x.left
    else x = x.right
z.p = y                 // väli p on vanema aadress
if y == NIL
    T.root = z          // puu T oli tühi
elseif z.key < y.key
    y.left = z
else y.right = z
```

Suurim ja väikseim võti

Kus paiknevad kahendotsingupuus minimaalne ja maksimaalne võti?

Kas oskaksite kirjutada selle kohta algoritmi, nii et puus liikumine lõppeb vastavalt suurima või väiksema võtmega tipu juures?

Mis on nende algoritmide keerukusklass?

Tipu kustutamine

Tipu kustutamiseks tuleb tipp leida, seejärel:

- kui tipul pole lapsi, kustutada
- kui tipul on üks laps, paigutatakse see kustutatud tipu asemele
- kui tipul on mõlemad lapsed, otsitakse tema asemele talle suuruselt järgnev tipp x, mille vasak alampuu on tühi;
  - suuruselt järgmine tipp paikneb kustutatava tipu paremas alampuus kõige vasakul.

Kahendotsingupuu efektiivsus

Otsimise, lisamise ja kustutamise keerukuse suurusjärk on logaritmiline, klass **O(log N)**. Miks?

Logaritmiline keerukus on otsimisel ja lisamisel tagatud siis, kui teede pikkused juurest lehtedeni on enam-vähem võrdsed.

Puu "kuju" sõltub lisatavate kirjete järjekorrast.

Millised puud tekivad, kui lisada samad võtmed erinevalt: 2, 1, 3 või 3, 1, 2 või 1, 2, 3?

Millises järjekorras kirjete lisamise puhul tekib otsimise jaoks ebaefektiivne puu?

AVL-puu

Otsingu parema efektiivsuse tagamiseks on vaja puud korrastada.

Otsingupuu efektiivemaks muutmise võtteid on mitmeid: nt AVL-puu, puna-mustpuu.

AVL-puu reegel ütleb, et tema iga alampuu jaoks vasaku ja parema alampuu kõrguste vahe ei tohi olla suurem kui üks.

AVL-puu "leiutasid" G. Adelson-Velsky ja E. Landis.

Tasakaalustamine (1)

Tasakaalufaktori abil iseloomustatakse tipu vasaku ja parema alampuu kõrguste vahet

Tasakaalufaktoril on kolm erinevat väärtust:

- - tipu vasak alampuu on 1 võrra kõrgem (-1)
- + tipu parem alampuu on 1 võrra kõrgem (+1)
- * tipu mõlema alampuu kõrgused on võrdsed (0)

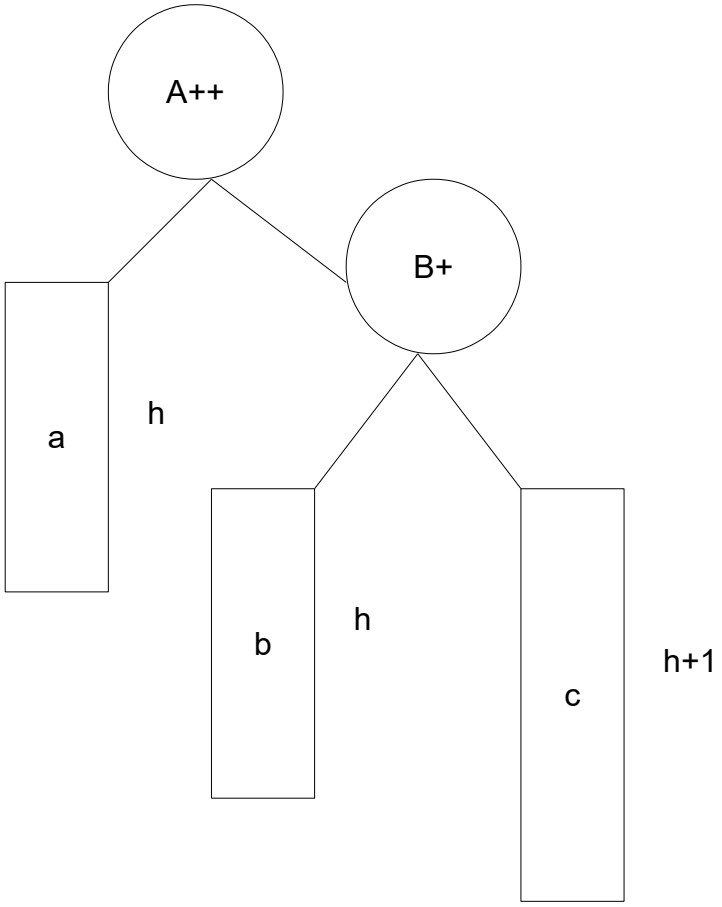
Tasakaalustamine.Olukord 1

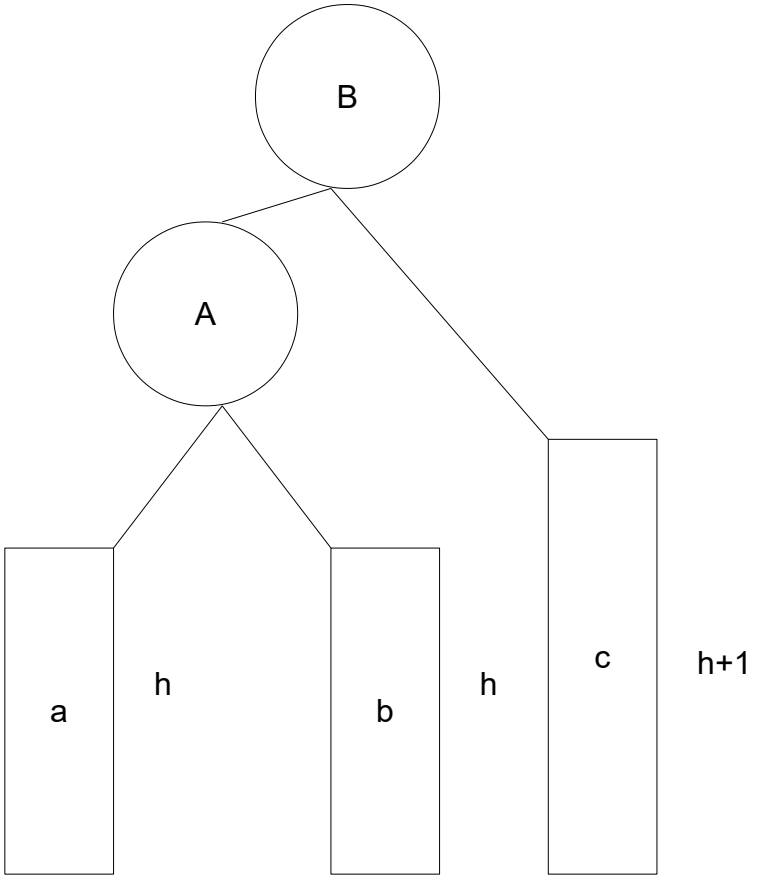
Tipul A on juba ühe võrra kõrgem parem alampuu (faktor +), parema alampuu (millel juureks B) paremasse alampuusse lisatakse tipp, sellega muutub ka tipu B faktor + ja tipus A on puu tasakaalust väljas (++).

Peegeldus: tipu A faktor on -, tema vasakusse alampuusse juurega B lisatakse vasakule juurde tipp. Selle tagajärjel saab tipu B faktoriks - ja tipus A on puu tasakaalust väljas (--).

Tasakaalustamiseks tehakse pööre, mille käigus tipud oma asukohti vahetavad (vt järgmine slaid).

Tasakaalustamine.Olukord 1





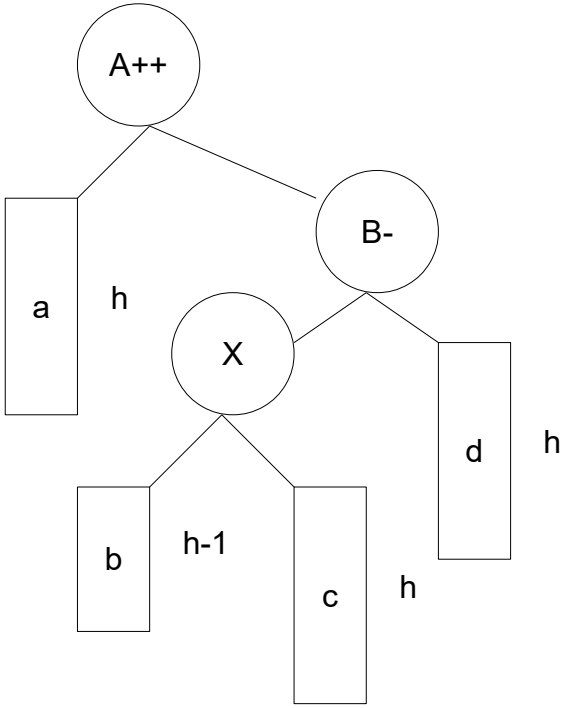
Tasakaalustamine.Olukord 2

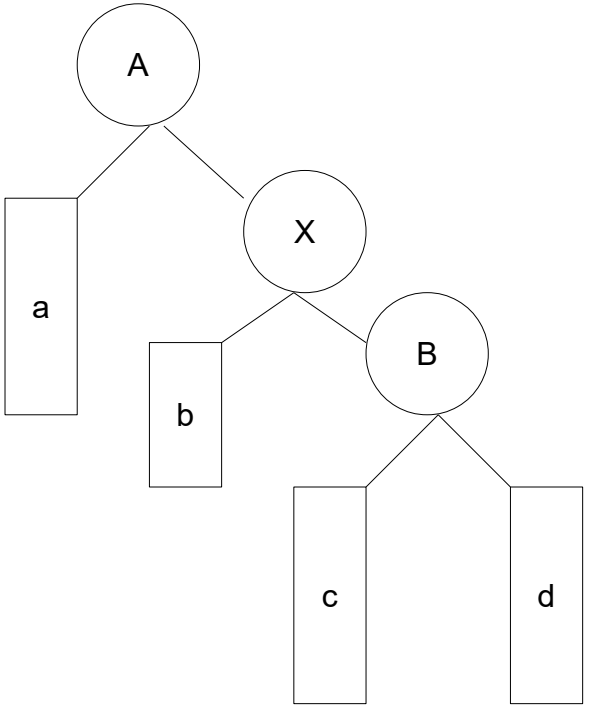
Tipul A on ühe võrra kõrgem parem alampuu (faktor +), tema paremasse alampuuse tipuga B lisatakse vasakule poole uus tipp, nii et tipu B jaoks tasakaalufaktor muutub – ning tipul A ++.

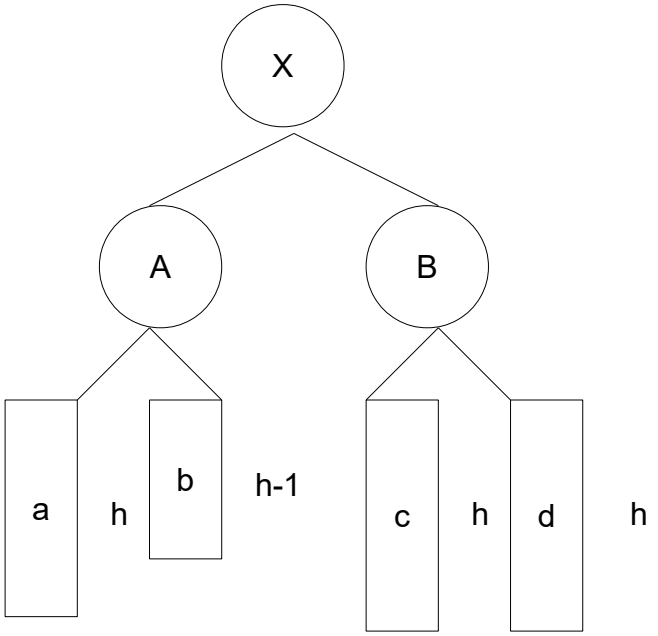
Peegeldus: tipul A on faktor - , tema vasakusse alampuusse tipuga B lisatakse paremale juurde tipp, mille tulemusena tipu B faktoriks saab + ning puu on tipus A tasakaalust väljas (--).

Tasakaalustamiseks tehakse 2 pööret (vt järgmine slaid)

Tasakaalustamine.Olukord 2







Ülesanne

Vaatame uuesti tuttavaid nimesid. Teeme nendest AVL-puu.

Nimed lisame järgmises järjekorras:

Kristi, Mihkel, Peeter, Carol, Kemal, Margit, Martin, Triinu

Puna-must puu

Puna-must puu *(Red Black Tree)* on teine meetod kahendotsingupuu tasakaalustamiseks.

Puu ei ole nii hästi tasakaalus kui AVL-puu, kuid tema tasakaalustamine on vähem töömahukas ja efektiivsus pole AVL-puust oluliselt halvem.

Jälgides tippude lisamisel ja kustutamisel värvide teatud skeemi ja tehes puus vajalikke tippude ümberpaigutusi, saab säilitada puu vastavuse puna-musta puu reeglitele, millega tagatakse puu tasakaalustatus.

Puna-musta puu reeglid

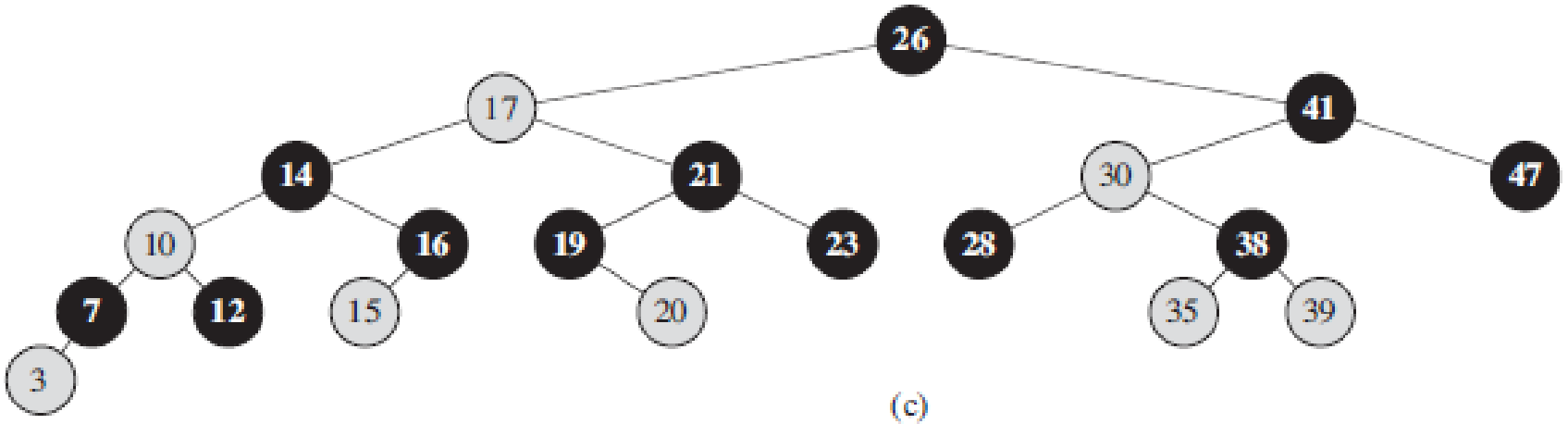
Reeglid on järgmised:

- Iga tipp on kas punane või must.
- Juur on must.
- Lehtede NULL-viidad (tühjad alampuud) on mustad.
- Kui tipp on punane, on mõlemad tema lapsed mustad.
- Iga tipu jaoks teed tipust järgnevate lehtedeni sisaldavad võrdselt musti tippe.

Puna-musta puu näide

Allikas: Cormen et al, "Introduction to algorithms", Third Edition.

Punased tipud on pildil hallid.



Hinnang efektiivsusele

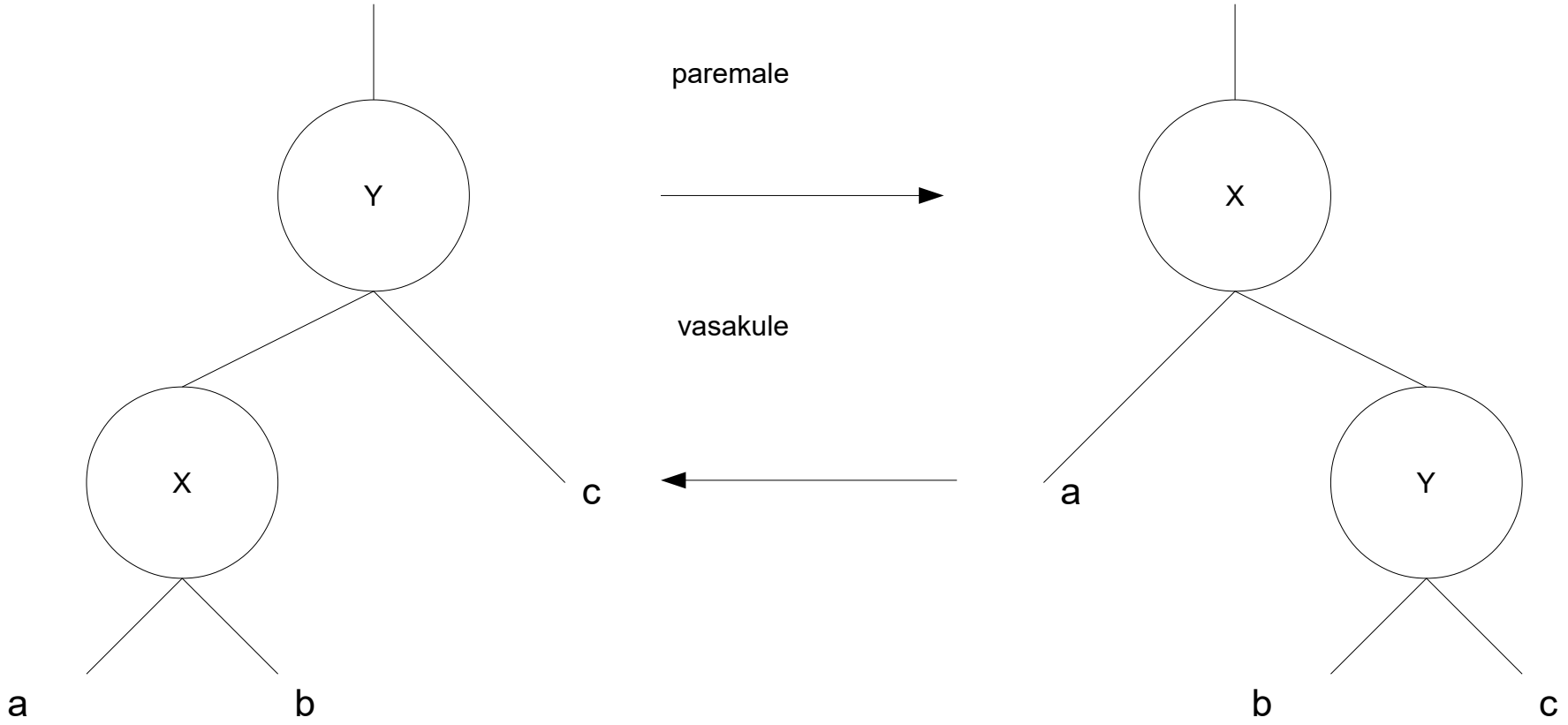
- Kui mustal tipul võib olla ka must järglane või vanem, siis punasel tipul tohivad olla vaid mustad lapsed.
- Seega kui reeglid on täidetud, pole ükski tee juurest leheni üle kahe korra pikem kui mistahes teine tee.
- Tänu kirjeldatud omadustele tehakse otsimisel N tipuga puna-mustas puus maksimaalselt `2*log(N+1)` sammu.

Puu korrastamine

Puu **korrastamiseks** kasutatakse kolme operatsiooni:

- **tippude värvimine** - punane tipp värvitakse mustaks ja must punaseks;
- **pööre vasakule** - tipu X parem laps saab uueks (alam)puu juureks ning X ise satub tema vasakuks lapseks;
- **pööre paremale** - tipu X vasak laps saab uueks (alam)puu juureks ning X ise satub tema paremaks lapseks.

Pöörded puus



Puu korrastamine

Lisatud tipp on alguses punane ja kui ta lisatakse punase tipu külge, on puu reeglid rikutud. Tekib kolm varianti:

1. Lisatud tipu isa ja onu on punased: lisatud tipu isal, onul ja vanaisal muudetakse värv (vanaisa punaseks, isa ja onu mustaks).

2. Lisatud tipu isa on punane ja onu on must. Lisatud tipp on oma isale paremaks lapseks, isa aga vasakuks lapseks. Reegleid rikkuva tipu ja tema punase vanema juures tehakse pööre vasakule. Tavaliselt viib see 3. juhuse tekkimisele.

3. Lisatud tipu isa on punane ja onu on must. Nii lisatud tipp kui ka isa on vasakud lapsed. Kõigepealt tehakse pööre paremale punase isa ja musta vanaisa suhtes. Seejärel värvitakse ringi punane isa ja must vanaisa.

Link mängimiseks

http://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Lisame arvud: 50 - 25 - 15 - 10 - 12

B-puu

B-puu *(B-tree)* on otsingupuu, mis kasutab kahendotsingupuu ideed (väiksemad vasakule, suuremad paremale) võtmete paigutamisel.

Igal tipul võib olla rohkem kui kaks last ning igas tipus võib olla rohkem kui üks võti, mis on järjestatud.

Tipus olevad võtmed on kui eraldajad tema alampuudes paiknevatele võtmetele.

Kas B-puu mõiste tuleb tuttav ette?

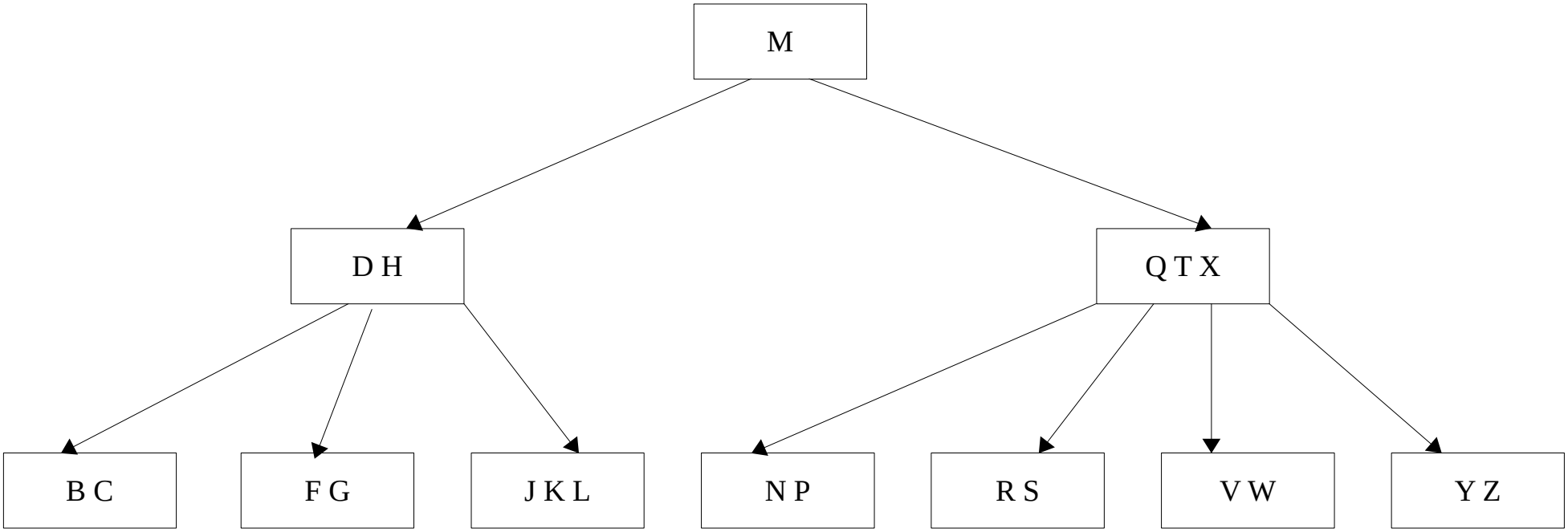
B-puu

Puu kõrgust hoitakse võimalikult väiksena ja ühes tipus on rohkem infot.

On optimeeritud suuremate andmeplokkide lugemiseks ja kirjutamiseks, sobides nii välismäluga suhtlemiseks.

Kasutatav näiteks failisüsteemides (nt NTFS) ja andmebaasides (indeksid).

Näide (kaashäälikud B-puus)



B-puu reeglid (1)

1. Igas tipus `x` on väljad, milles hoitakse:
   - tipu võtmete arvu `x.n`,
   - võtmeid mittekahanevas järjekorras:

     $x.k_1 <= x.k_2 <= \ldots <= x.k_n$,

   - tähist, kas tipp on leht
2. Kui `x` on sisemine tipp, on temas `x.n` elementi ja `x.n+1` viita lastele
3. Tipus olevad võtmed (`x.n` tükki) on piirideks, mille järgi jaotatakse võtmed alampuude vahel.

B-puu reeglid (2)

4. Kõik lehed on samal tasemel

5. Tipus olevate võtmete arvule on määratud maks ja min väärtuse. See on ühesugune kogu `t>=2` järku puu tippude jaoks ehk `t` on B-puu **minimaalne järk** *(minimum degree):*

- igas tipus (va juur) on vähemalt `t-1` võtit, st sisemistel tippudel on vähemalt `t` last.
- igas tipus ei ole rohkem kui `2t-1` võtit, st sisemisel tipul ei ole rohkem kui `2t` last. Tipp on täis *(full)* kui temas on `2t-1` võtit

2-3-4-puu

2-3-4-puu *(2-3-4 tree)* on B-puu, kus t = 2.

Omadused:

- Igal tipul võib olla 2 kuni 4 last (ja vastavalt igas tipus on 1 kuni 3 võtit).
- Kõik lehed on samal tasemel
- Võtmed on puus sorteeritult.
- Kasutatakse näiteks sõnastike *(dictionary)* realiseerimiseks.