

4.2 Fibonacci-kasat

Fibonacci-kasoilla voidaan toteuttaa samat operaatiot kuin binomikasoilla.

Pääsiallinen ero on, että paljon Decrease-Key-operaatioita sisältävät jonot nopeutuvat.

- **Primin algoritmi** pienimmälle virittävälle puulle nopeutuu ajasta $O(|E| \log |V|)$ aikaan $O(|E| + |V| \log |V|)$ (sivuutetaan tällä kurssilla; vrt. Kruskalin algoritmi $O(|E| \log |V|)$)
- samoin Dijkstran algoritmi nopeutuu ajasta $O(|E| \log |V|)$ aikaan $O(|E| + |V| \log |V|)$

Fibonacci-kasojen **vakiokertoimet ovat suuret**, mutta huolellisesti toteutettuna ne nopeuttavat algoritmeja myös käytännössä suurilla aineistoilla.

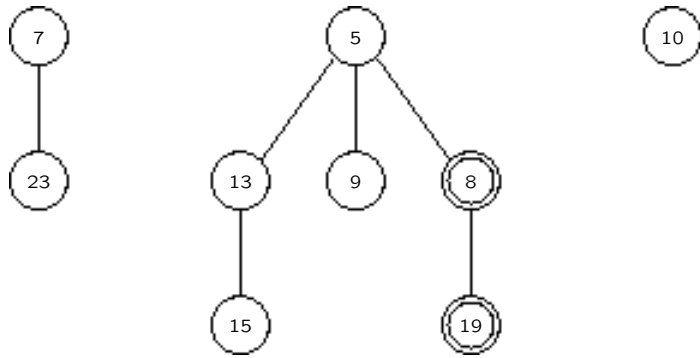
Myös **toteuttaminen on hankalaa** (mutta valmiita toteutuksia on saatavilla, kuten kaikille muillekin perustietorakenteille).

Fibonacci-kasan talletusrakenne on joukko järjestämättömiä puita.

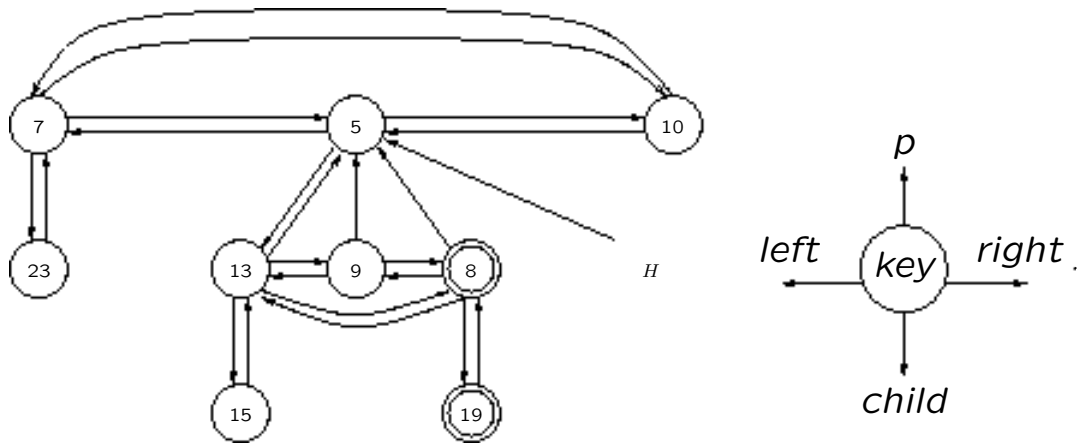
- puiden juuret linkitetty yhdeksi *juurilistaksi*
- kunkin solmun lapset on linkitetty yhdeksi listaksi
- lapsista on linkki vanhempaansa
- kukin puu toteuttaa kasaehdon (vanhemman avain ei suurempi kuin lapsen avain)
- kasa esitetään osoittimella juurilistan pienimpään alkioon (eli koko kasan pienimpään alkioon)

Kaikki listat ovat kahteen suuntaan linkitettyjä rengaslistoja.

Kussakin solmussa on lisäksi tieto sen lasten lukumäärästä (*degree*) sekä [merkki](#), joka voi olla tosi tai epätosi.



Esimerkki Fibonacci-kasasta; merkityt solmut rengastettu



Fibonacci-kasan linkitetty rakenne

Järjestämätön binomipuu on kuin järjestetty binomipuu, mutta solmun lasten järjestys ei ole merkitsevä. Näillä on luonnollisesti vastaavat ominaisuudet kuin järjestetyillä, erityisesti n -alkioisen puun solmun aste on korkeintaan $\log n$.

Jos kasan kaikki puut ovat binomipuita, niin operaatiot Insert, Minimum, Extract-Min ja Union pitävät tämän voimassa.

Binomipuista poiketen kuitenkin Union ja Insert voivat johtaa tilanteeseen, jossa kasassa on useita saman kokoisia binomipuita. Vasta Extract-Min tiivistää kasan sellaiseksi, että jokaisella juurilistan solmulla on eri asteluku.

Operaatiot Decrease-Key ja Delete voivat muuttaa binomipuita ei-binomipuiksi. Solmuissa olevia merkkejä käytetään osoittamaan tällaiset rakennepoikkeamat.

Tasoitettussa analyysissä kasan H potentiaali on

$$\Phi(H) = at(H) + 2am(H)$$

missä $t(H)$ on juurilistan pituus, $m(H)$ merkittyjen solmujen lukumäärä ja $a > 0$ sopiva vakio. Siis operaatioita "palkitaan" juurilistan lyhentämisestä ja merkkien poistamisesta, jotka tekevät rakenteen "binomikasamaisemmaksi".

Kasajoukon potentiaali on yksittäisten kasojen potentiaalien summa.

Binomipuista tiedettiin, että jos solmun x asteluku on k ja sen lapset oikealta vasemmalle y_1, \dots, y_k , niin solmun y_i asteluku on $i - 1$.

Fibonacci-kasoissa on voimassa seuraava heikompi ominaisuus:

Propositio Olkoon solmun x asteluku $\text{degree}[x] = k$ ja sen lapset y_1, \dots, y_k siinä järjestyksessä, jossa ne on liitetty solmuun x (vanhin lapsi ensin). Nyt $\text{degree}[y_i] \geq i - 2$ kaikilla i .

Operaatioiden toteutuksen yhteydessä tarkastelemme, miten tämä pidetään voimassa.

Perusidea lyhyesti on, että solmujen linkittämisessä yhdistetään aina samanasteisia solmuja (kuten binomikasoissa). Siis kun y_i linkitetään solmun x lapseksi, on voimassa $\text{degree}[y_i] = \text{degree}[x] = i - 1$.

Myöhemmin solmun y_i sallitaan menettää *yksi* lapsi. Jos solmu y_i menettäisi toisenkin lapsen, rakenne järjestellään uudelleen.

Järjestelytarpeen havaitsemiseksi käytetään merkkejä:

solmussa y on merkki, jos y on jonkin solmun x lapsi ja y on **menettänyt yhden lapsen** tultuaan solmun x lapseksi.

Olkoon f_k k . Fibonacci luku: $f_0 = 0$, $f_1 = 1$,
 $f_{n+2} = f_{n+1} + f_n$. Tunnetusti (helppo induktio)

$$f_{k+2} \geq 1 + \sum_{i=0}^k f_i.$$

Olkoon $\text{size}(x)$ sen alipuun solmujen lukumäärä, jonka juuri on x , ja olkoon $\text{size}(k)$ pienin mahdollinen $\text{size}(x)$ kun $\text{degree}[x] = k$.

Kun edellinen propositio oletetaan todeksi, saadaan nyt

Lause $\text{size}(k) \geq f_{k+2} \geq \phi^k$ missä $\phi = (1 + \sqrt{5})/2$.

Olkoon $D(n)$ suurin solmun aste n -solmuisessa Fibonacci-kasassa.

Korollaari $D(n) \leq \log_{\phi} n = O(\log n)$

Tämä korollaari korvaa Fibonacci-kasoilla binomikasojen tarkemman arvion $D(n) = \lfloor \log n \rfloor$.

Lauseen todistus Väite ilmeisesti pätee kun $k \leq 1$.

Olkoon nyt x solmu jonka aste on $k \geq 2$ ja lapset ikäjärjestyksessä y_1, \dots, y_k . Tehdään induktio-oletus $\text{size}(j) \geq f_{j+2}$ kun $j < k$. Koska $\text{size}[y_1] = 1$, saadaan

$$\begin{aligned} \text{size}(x) &= 1 + \sum_{i=1}^k \text{size}(y_i) \\ &= 2 + \sum_{i=2}^k \text{size}(y_i) \\ &\geq 2 + \sum_{i=2}^k \text{size}(i-2) \\ &\geq 2 + \sum_{i=2}^k f_i \\ &= f_{k+2}. \end{aligned}$$

Koska x oli mielivaltainen, pätee $\text{size}(k) \geq f_{k+2}$.

Toinen epäyhtälö seuraa aiemmin johdetusta Fibonaccin lukujen suljetusta kaavasta (tai helpolla induktiolla). \square

Esitetään nyt itse operaatiot ja niiden tasoitettut aikavaatimukset. Yksinkertaisuuden vuoksi jätämme käsittemättä kasan tyhjenemiset ja muut helpot erikoistapaukset.

Jatkossa b merkitsee jotain vakiota, joka on valittu niin suureksi että kaikki "vakioajan" vievät toimitukset sujuvat ajassa b . (Lisäksi erikseen ei aina mainita, että kysymys on ylärajasta.)

Muistetaan, että kasojen H_1, \dots, H_m yhteispotentiaali on

$$\sum_{i=1}^m \Phi(H_i) = \sum_{i=1}^m (at(H_i) + 2am(H_i))$$

missä t on juurilistan pituus, m merkittyjen solmujen lukumäärä ja $a > 0$ myöhemmin valittava vakio.

Aluksi ei ole lainkaan kasoja, eli potentiaali on nolla. Selvästi potentiaali on aina ei-negatiivinen, joten tasoitettu aikavaativuus on todellisen aikavaativuuden yläraja.

Make-Fib-Heap: Todellinen aikavaativuus on vakio eli kork. b . Luodulla uudella puulla potentiaali on nolla, joten potentiaalissa ei muutosta.

Fib-Heap-Insert(H, x): Toteutetaan tekemällä alkiosta x yhden solmun puu ja liittämällä kasan H juurilistaan. Todellinen aikavaativuus b , potentiaalinen muutos a , joten tasoitettu aikavaativuus $a + b$.

Fib-Heap-Minimum: Todellinen aika triviaalisti vakio, koska kasa esitetään osoittimena minimialkioon. Potentiaali ei muutu, joten tasoitettu aikavaativuus b .

Fib-Heap-Union(H_1, H_2): Yhdistetään kasojen H_1 ja H_2 juurilistat (pelkkä konkatenatio, ei mitään uudelleenjärjestelyjä). Palautetaan osoitin joko H_1 :n tai H_2 :n minimialkioon, sen mukaan kumpi on pienempi. Todellinen aikavaativuus b , yhteenlaskettu potentiaali ei muutu.

Fib-Heap-Extract-Min(H): periaate on seuraava:

1. $z := \min(H)$
2. $H' :=$ solmun z lapsilista
3. poista z kasan H juurilistasta
4. yhdistä H' kasan H juurilistaan
5. tiivistä kasa H

Kuten edellä, askelessa 4 vain konkatenoidaan listat.

Askelessa 5 tiivistetään kasan H uusi juurilista sellaiseksi, että kullakin juurella on eri asteluku (vrt. binomikasa).

Tiivistämisessä puita yhdistetään seuraavalla proseduurilla:

Fib-Heap-Link(H, y, x):

1. poista y kasan H juurilistasta
2. linkitä y solmun x lapseksi
3. $degree[x] := degree[x] + 1$
4. $mark[y] := \text{False}$

Tiivistäminen tapahtuu seuraavalla proseduurilla:

Consolidate(H):

1. **for** $d := 0$ **to** $D(n)$ **do** $A[d] := \text{Nil}$
2. **for** kaikilla kasan H juurilistan solmuilla x **do**
3. $d := \text{degree}[x]$
4. **while** $A[d] \neq \text{Nil}$ **do**
 % *tallessa on jo yksi d -asteinen juuri*
5. **if** $\text{key}[x] > \text{key}[A[d]]$ **then** vaihda x ja $A[d]$
6. Fib-Heap-Link($H, A[d], x$)
7. $A[d] := \text{Nil}$
8. $d := d + 1$
9. $A[d] := x$
10. muodosta lista niistä alkoista $A[d]$ jotka eivät ole Nil
11. $H :=$ osoitin listan pienimpään alkioon

Kullakin x siis katsotaan, onko asteluvun $d = \text{degree}[x]$ mukainen paikka $A[d]$ vapaana taulukossa A , joka on aluksi tyhjä.

Jos paikka on vapaa, x talletetaan siihen ja siirrytään seuraavaan juurilistan alkioon.

Jos paikka ei ole vapaa vaan siellä on juuri y , linkitetään y ja x ja saadaan uusi astetta $d + 1$ oleva puu. Paikka $A[d]$ vapautuu, ja seuraavaksi katsotaan onko yhdistetylle puulle vapaana paikka $A[d + 1]$.

Huomaa, että linkitettäessä y ja x pätee aina $\text{degree}[x] = \text{degree}[y]$ (vrt. sivun 221 propositio).

Kaikki muu paitsi Consolidate-proseduurin **for**-silmukka (rivit 2–9) menee selvästi ajassa $O(D(n))$.

Consolidate(H)-kutsun **for**-silmukan aikavaativuus on $O(t(H) + D(n))$, koska juurilistassa on aluksi $t(H)$ solmua ja siihen lisätään korkeintaan $D(n)$ solmun z lasta. Siis koko proseduurin Fib-Heap-Extract-Min todellinen aikavaatimus on $b(D(n) + t(H))$.

Juurilistan pituus on aluksi $t(H)$ ja lopuksi korkeintaan $D(n) + 1$. Merkattuja solmuja ei tule ainakaan lisää. Potentiaalın muutos on siis

$$\begin{aligned}\Delta\Phi(H) &\leq a(D(n) + 1 + 2m(H)) - a(t(H) + 2m(H)) \\ &= a(D(n) + 1 - t(H)).\end{aligned}$$

Tasoitetuksi aikavaativuudeksi tulee siis

$$b(D(n) + t(H)) + a(D(n) + 1 - t(H)) \leq (2a + b)D(n)$$

olettaen että valitaan $a \geq b$ (ja $D(n) \geq 1$).

Fib-Heap-Decrease-Key(H, x, k): Tässä siis vaaditaan $k < key[x]$.

Jos avaimen $key[x]$ pienentäminen rikkoo kasaominaisuuden, ratkaistaan ongelma yksinkertaisesti ottamalla x juurilistalle (proseduuri Cut).

Tällöin solmun x vanhempi y menettää yhden lapsen.

Jos y ei vielä ole merkattu, tilanne korjautuu merkkamalla y . Jos y on jo merkattu (ts. y on jo menettänyt yhden lapsen), vaaditaan isompi korjaustoimi: solmu y viedään sekin juurilistalle. Nyt solmun y vanhempi puolestaan menettää lapsen, joten korjaamista voidaan joutua jatkamaan rekursiivisesti (proseduuri Cascading-Cut).

Perusproseduuri on siis seuraava:

Fib-Heap-Decrease-Key(H, x, k):

$key[x] := k$

if x ei ole juuri **then**

$y := p[x]$

if $key[x] < key[y]$ **then**

Cut(H, x, y)

Cascading-Cut(H, y)

if $key[x] < key[\min(H)]$ **then** $\min(H) := x$

Apuproseduurit ovat seuraavat:

$\text{Cut}(H, y, x)$:

poista x solmun y lapsilistasta
 $\text{degree}[y] := \text{degree}[y] - 1$
liitä x kasan H juurilistaan
 $\text{mark}[x] := \text{False}$

$\text{Cascading-Cut}(H, y)$:

if y ei ole juuri **then**
 if $\text{mark}[y] = \text{False}$
 then $\text{mark}[y] := \text{True}$
 else
 $\text{Cut}(H, y, z)$
 $\text{Cascading-Cut}(H, z)$

Todellinen aikavaativuus on selvästi $(Z + 1)b$, missä Z on suoritettavien Cascading-Cut-kutsujen lukumäärä.

Juurilistan pituus on aluksi $t(H)$ ja lopuksi $t(H) + Z$, sillä x tulee juureksi ja viimeistä lukuunottamatta jokainen Cascading-Cut lisää yhden juuren.

Viimeistä lukuunottamatta jokainen Cascading-Cut poistaa yhden merkin. Viimeinen voi lisätä yhden merkin. Siis lopuksi merkkejä on korkeintaan $m(H) - Z + 2$.

Tasoitetuksi kustannukseksi tulee

$$b(Z + 1) + aZ + 2a(2 - Z) \leq 4a + b$$

olettaen, että valitaan $a \geq b$.

Huomataan, että merkityn solmun kahdesta "potentiaaliyksiköstä" toinen maksaa sen leikkaamisesta tulevat kustannukset ja toinen jää solmuun koska siitä tulee juuri.

Solmun poistaminen palautuu jo nähtyyn:

Fib-Heap-Delete(H, x):

Fib-Heap-Decrease-Key($H, x, -\infty$)

Fib-Heap-Extract-Min(H)

Tasoitettu aikavaativuus on siis $O(D(n))$.

Yhteenveto: Olemme saaneet tasoitetuiksi aikavaativuuksiksi $O(1)$ operaatioille Make-Heap, Insert, Union, Minimum ja Decrease-Key, ja $O(D(n))$ operaatioille Extract-Min ja Delete.

Koska potentiaali ei koskaan laske alkuarvonsa ali, nämä ovat ylärajat myös todellisille kustannuksille operaatiota kohti koko operaatiojonon yli yhteenlaskettuna.

Kun vielä toteamme, että alussa esitetty propositio tosiaan pitää paikkansa, tiedämme että $D(n) = O(\log n)$ eli saamme halutut aikavaativuudet.

Propositio Olkoon solmun x asteluku $\text{degree}[x] = k$ ja sen lapset y_1, \dots, y_k siinä järjestyksessä, jossa ne on liitetty solmuun x (vanhin lapsi ensin). Nyt $\text{degree}[y_i] \geq i - 2$ kaikilla i .

Todistus Ainoa tilanne, jossa solmu y liitetään solmun x lapseksi, on proseduurissa Consolidate, ja tällöin aina $\text{degree}[x] = \text{degree}[y]$.

Siis jos y_i on solmun x nykyisistä lapsista aikajärjestykseltään numero i , niin solmua y_i solmuun x linkitettäessä solmun y_i asteluku oli $i - 1$.

Sinä aikana kun y_i on ollut solmun x lapsi, y_i on voinut menettää korkeintaan yhden lapsen. Nimittäin jos se olisi menettänyt toisenkin lapsen, se olisi leikattu ja siitä olisi tullut juuri.

Siis edelleen pätee $\text{degree}[y_i] \geq i - 2$. \square