Anna Hietanen
Tiralabra
10/7/12

Määrittelydokumentti

1. Mitä algoritmeja ja tietorakenteita toteutat työssäsi

Aion toteuttaa työssäni ainakin kaksi, mahdollisesti kolme kekotietorakennetta: binomikeon, binäärikeon ja fibonaccikeon.

2. Mitä ongelmaa ratkaiset ja miksi valitsit kyseiset algoritmit/tietorakenteet

Esittelen ja vertailen kolmea eri kekoa ja niiden rakenteita. Valitsin aiheen, koska en ole kovin taitava koodaaja, mutta aihe tuntui kuitenkin hallittavalta, koska kekorakenteen ovat erillisiä.

3. Mitä syötteitä ohjelma saa ja miten näitä käytetään

Ohjelma itsessään ei saa syötteitä, mulla keoilla on monta eri käyttötarkoitusta mm. verkkoalgoritmit käyttävät kekoja solmujen läpikäyntijärjestyksen tallentamiseen ja selvittämiseen.

4. Tavoitteena olevat aika- ja tilavaativuudet (m.m. O-analyysi)

Binomikeko

Merge() menee läpi ajassa O(log n). Insert() -metodi menisi itsessään läpi vakioajassa, sillä siinä tehdään vain uusi keko ja yhdistetään se vanhaan kekoon. Merge() kuitenkin aiheuttaa sen, että insertin aikavaativuus on O(log n). Pienimmän alkion löytäminen ja poistaminen tapahtuu ajassa O(log n). Mikäli pienintä alkiota on ylläpidetty pointterilla, sen löytäminen tapahtuu ajassa O(1).

Binäärikeko

Binäärikeossa metodit insert() ja delete() menevät läpi parhaimmassa tapauksessa ajassa O(log n). Delete() tapahtuu myös vakioajassa, eli on yhtä suuri kuin puun korkeus. Keko rakentuu ajassa O(n log n), sillä jokainen insert() vie aikaa O(log n) ja niitä tehdään n kertaa, niin kauan kuin lisättäviä alkioita on.

Fibonaccikeko

Fibonaccikeon tavoitteena olisi suorittaa insert nopeammin kuin em. keoissa. Delete_min on suurinpiirtein yhtätehokas kuin muissakin, koska se tapahtuu ajassa O(log n).

Toteutus:

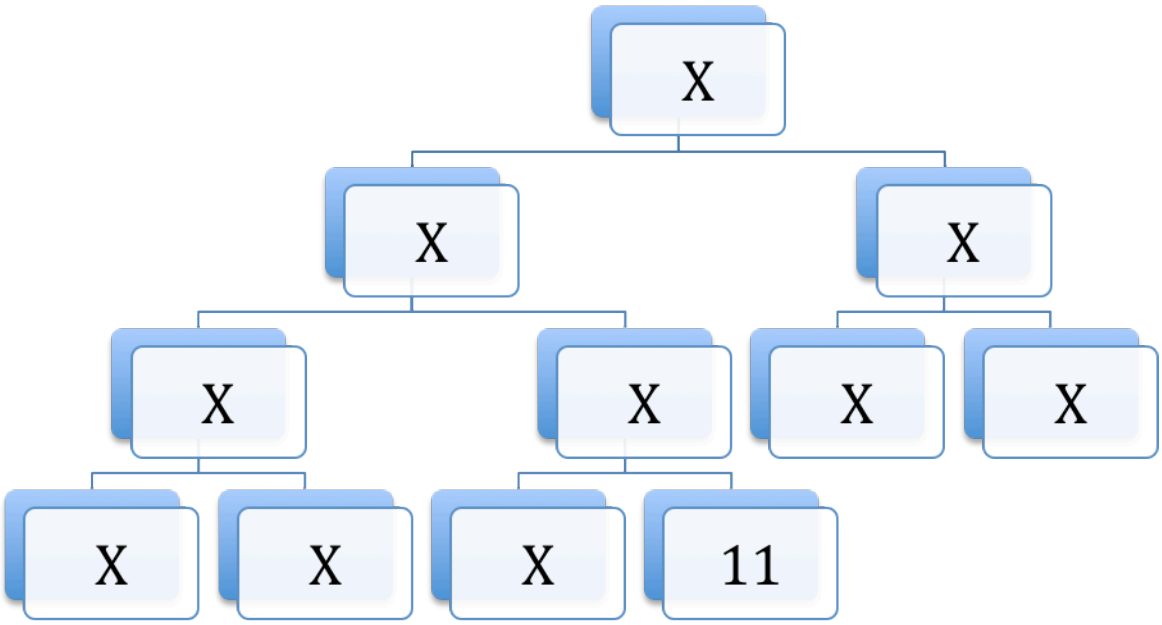
1. Ohjelman yleisrakenne

Vertailen kolmea eri kekorakennetta.

Anna Hietanen
Tiralabra
10/7/12

Binäärikeko:

Insert() -metodissa muutetaan ensin luku binäärimuotoon, sitten aloitetaan binääriluvun läpikäynti toisesta numerosta siirtymällä puussa vasemmalle jos numero on 0 ja oikealle, jos numero on 1. Esimerkiksi luku 11 on binäärinä 1011. Nyt siis puussa liikuttaisiin vasen -> oikea -> oikea.



Delete taas toimii niin, että juuri korvataan solmulla, joka on lisätty viimeiseksi. Tämä etsitään samalla tavalla kuin insertissä. Insertin ja deleten jälkeen siirretään solmu oikeaan paikkaan kutsumalla aina heap-up:ia tai heap-down:ia, jotka varmistavat, että minimikeon ehdot täyttyvät.

Binomikeko:

Binomikeko koostuu useammasta puusta, jotka ovat kooltaan 2^n (n kuuluu luonnollisiin lukuihin). Binomikeossa lisäys tapahtuu tekemällä uudesta solmusta oma keko ja yhdistämällä se jo olemasssaolevan keon kanssa. Kun havaitaan, että keossa on kaksi samankokoista alipuuta, ne yhdistetään merge() -metodin avulla. Pienimmän alkion poistamisessa aluksi pienin alkio, joka löytyy keon jostakin juuresta, poistetaan juurilistasta. Jos kyseisellä juurella on lapsia, järjestys käännetään, lapset lisätään uuteen kekoon ja uusi keko yhdistetään jo olemassaolevan keon kanssa. Pienin alkio etsitään find_min() -metodilla, joka käy juurilistan läpi ja etsii niistä pienimmän alkion. Binomikeon toteutuksessa löytyy toString-metodi, joka helpottaa keon hahmottamista tulostamalla juurilistan ja lapset eri kerroksissa.

Fibonaccikeko:

Fibonaccikeon insert on yksinkertainen: alkiot vain lisätään kahteen suuntaan linkitettyyn listaan oikeaan paikkaan, ja tarkastetaan olisiko lisättävä pienempi kuin tämänhetkinen minimi. Pienintä alkiota poistettaessa poistetaan ensiksi pienimmän arvon omaava alkio juurilistasta, ja sen lapsista tulee uusia juuria uusille alipuille. Decrease key vähentää yhden alkion arvoa, mikäli parametrinä on pienempi arvo. Jos kekoehto vioittuu eli alkion vanhemmalla on pienempi arvo, leikataan solmu vanhemmasta. Jos vanhempi ei ole juuri, se merkitään. Jos se on jo merkitty, se myös se leikataan ja sen vanhempi merkitään. Tätä jatketaan puussa ylöspäin, kunnes päästään merkitsemättömään solmuun. Delete vähentää solmun arvon mahdollisimman pieneksi, jolloin siitä tulee koko keon pienin. Sitten kutsutaan pienimmän alkion poistoa.

2. Saavutetut aika- ja tilavaativuudet (m.m. O-analyysi pseudokoodista)

Määrittelydokumentin aikavaativuudet on saavutettu.

3. Suorituskyky- ja O-analyysivertailu (mikäli työ vertailupainotteinen)

Vertailussa keskeisimmät metodit.

	Binäärikeko	Binomikeko	Fibonaccikeko
insert	O(log n)	O(log n)	O(1)
deleteMin	O(log n)	O(log n)	O(log n) amortisoitu
decreaseKey	O(log n)	O(log n)	O(1)
100 000 lisäys	160ms	80ms	22ms
100 000 poisto	140ms	35ms	124ms
käänteinen lisäys	95ms	60ms	44ms
Käänteinen poisto	65ms	18ms	158ms

Binaarikeko:

Aikaa kului 100000 alkion lisäykseen: 160.0ms.
Aikaa kului 1000000 alkion lisäykseen: 1642.0ms.
Aikaa 100000 alkion poistamiseen kului: 140.0ms.

Käänteinen 100000 alkion (100000, 99999... 1, 0) lisäys vei aikaa 95.0ms.
Käänteinen 100000 alkion poistaminen kesti 65.0ms.

Binomikeko:

Aikaa kului 100000 alkion lisäykseen: 80.0ms.

Aikaa kului 1000000 alkion lisäykseen: 792.0ms
Aikaa 100000 alkion poistamiseen kului: 35.0ms.

Käänteinen 100000 alkion (100000, 99999... 1, 0) lisäys vei aikaa 60.0ms.
Käänteinen 100000 alkion poistaminen kesti 18.0ms.

Fibonaccikeko:

Aikaa kului 100000 alkion lisäykseen: 22.0ms.
Aikaa kului 1000000 alkion lisäykseen: 490.0ms.
Aikaa 100000 alkion poistamiseen kului: 124.0ms.

Käänteinen 100000 alkion lisäys vei aikaa 44.0ms.
Käänteinen 100000 alkion poistaminen kesti 158.0ms.

4. Työn mahdolliset puutteet ja parannusehdotukset

5. Lähteet

http://en.wikipedia.org/wiki/Binary_heap
http://en.wikipedia.org/wiki/Binomial_heap
http://en.wikipedia.org/wiki/Fibonacci_heap
http://www.cs.helsinki.fi/group/java/k11/tira/tira2011.pdf
http://www.cs.helsinki.fi/u/mnykanen/ATPe/luennot3.pdf
http://en.wikipedia.org/wiki/Heap_(data_structure)
T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms, 3rd ed., MIT Press, 2009.

Anna Hietanen
Tiralabra
10/7/12

Käyttöohjeet:

1. Miten koodi käännetään? (jos valmis ajettava ohjelma ei mukana palautuksessa)

Jokainen keko on oma erillinen projektinsa. Algoritmit voi ajaa komentoriviltä käskyllä javac tiedosto.java, tai avata suoraan jollakin IDE:llä, esim NetBeans tai Eclipse.

2. Miten ohjelma suoritetaan, miten eri toiminnallisuuksia käytetään?

Keoilla on eri metodeja, kuten alkioiden poistoa ja lisäystä. Kekoa käytetään luomalla siitä ilmentymä ja kutsumalla näitä metodeita.

Testaus:

1. Mitä on testattu, miten tämä tehtiin?

Testasin kolmen eri keon keskeisimpiä metodeita (insert ja delete) erikokoisilla syötteillä ja mittasin samalla kuluneen ajan.

2. Minkälaisilla syötteillä testaus tehtiin? (vertailupainotteisissa töissä tärkeätä)

Syötteenä käytin normaaleja integer-lukuja, ainoastaan syötteen koko vaihteli ja sen pohjalta sain selville kekojen eroavaisuudet. Kun yritin laittaa syötteen kooksi 10 miljoonaa, huomasin, että NetBeans ei voi enää ajaa testiä läpi, menee liian kauan. Miljoonan syötteen testi on siis suurin millä testattu. Mielenkiintoista olisi, jos voisi testata vielä paljon suuremmilla syötteillä ja selvittää niihin kuluvat ajat.

3. Miten testit voidaan toistaa?