Toteutusdokumentti

Tietorakenteet ja algoritmit harjoitustyö, Keko vertailut

Pirjo Turunen

Tietorakenteet ja algoritmit harjoitustyö

16.06.2013

Helsingin yliopisto, Tietojenkäsittelytieteen laitos

Kristiina Paloheimo, Mika Huttunen

JOHDANTO

Dokumentin aiheena on tietorakenteet ja algoritmit harjoitustyönä Javalla toteutettu kolmen eri keon toteutus ja niiden aika-ja tilavaativuuksien vertailu.

Toteutetut keot ovat dkeko, binomikeko sekä fibonaccikeko, joiden suorituskykyä vertaillaan prioriteettijonon tietorakenteena.

1. KUVAUS OHJELMAN RAKENTEESTA JA TOIMINNASTA

1.1 Ohjelman yleisrakenne

Package Kekoharjoitus koostuu 8 luokasta. Seuraavissa aliluvuissa kuvataan luokkien rakenne ja toiminta. Kekoharjoitus toteuttaa kolmella eri keko tietorakenteella minimikeon; Dkekona,binomikekona ja fibonaccinkekona.

1.1.1 class Kekoharjoitus

Sisältää main funktion. Vastuussa sovelluksen käynnistyksestä ja sulkemisesta. Sisältää joukon suorituskyky testejä, joissa verrataan eri kekojen suorituskykyä kaikille keoille yhteisissä operaatioissa; insert,makeHeap,findMin,deleteMin , decrease ja merge. Sovelluksessa ei ole toteutettu erillistä käyttöliittymää. Vertailun tulokset tulostetaan system.out:iin. Ko. testit eivät siis ole yksikkötestejä vaan yksikkötestit on toteutettu erikseen erillisinä testiluokkina jokaista toteutettua varsinaista luokkaa kohti. Kekoharjoitus luokassa toteutetut testit on eritelty testausdokumentissa.

1.1.2 class Solmu

Minimikekojen perustietorakenne Solmu luokan value arvo vastaa minimikeon key arvoa. Minimikeko palauttaa pienimmän keon solmun, jonka Solmu.value arvo on pienin. Kaikkien kolmen kekototeutuksen minimi minimikeosta perustuu Solmu luokan value arvoon.

public Solmu(int value)	Konstruktorissa välitetään kokonaisluku, joka määrittää Solmu olion paikan minimikeossa.
public int getValue()	Funktio palauttaa Solmu olion value arvon
public void setValue(int value)	Funktio asettaa Solmu olion value arvon

1.1.3 class Dkeko

Dkeko tietorakenteena toteutettu minimikeko. Luokka toteuttaa perusoperaatiot dkeko tietorakenteesta siten kuin ne on yleisesti kirjallisuudessa määritelty. Perusoperaatiot ovat kuvattu alla, mutta luokka sisältää myös tukifunktioita, jotka on kuvattu JavaDocissa. Dkeko on toteutettu kaksisuuntaisena linkitettynä listana,jossa alkioina on Kekoalkio oliota, jotka muodostavat linkitetynlistan rakenteen jasisältävät Solmu oliot. Linkitetyn listan ensimmäiseen ja viimeiseen alkioon osoittaa Dkeon jäsenmuuttujat min ja tail.

public Dkeko(int d)	Konstruktorissa välitetään kokonaisluku, joka määrittää Dkeon haarautumisasteen.

public int decreaseKey(Kekoalkio alkio, int value)	Funktio pienentää keossa olevan alkion key arvoa, jos parametrina annettu value arvo on pienempi kuin keossa olemassaoleva arvo.Kekoehdon rikkoutuessa suoritetaan ylöspäin korjaus.Palauttaa 0, jos asetus onnistuis, muutoin palauttaa -1.
public Solmu deleteMin()	Funktio poistaa minimikeosta keon pienimmän alkion ja palauttaa sen.Keon viimeinen lehti nostetaan keossa ylimmäksi ja tarkastetaan senjälkeen ja korjataan mahdollinen kekoehdon rikkoutuminen alaspäin korjaamalla.
public Solmu findMin()	Funktio palauttaa minimikeon pienimmän alkion eli Solmu olion, jonka value arvo on keon pienin. Solmu säilyy keossa. Keon jäsenmuuttuja min osoittaa suoraan keon pienimpään alkioon.
public void insert(Solmu x)	Funktio lisää Solmu luokan olion minimikekoon, olion omistajuus siirtyy.Puu täytetään tasoittain ylhäältä alhaalle, vasemmalta oikealle.Kekoehdon rikkoutuessa suoritetaan korjaus ylöspäin.
public static Dkeko makeHeap(int d)	Luo uuden tyhjän dkeon,parametrina annetaan uuden keon haarautumisaste.
public static Dkeko merge(Dkeko t1, Dkeko t2)	Yhdistää kaksi dkekoa toisiinsa luomalla uuden keon ja yhdistää annettut keot toisiinsa lisäämällä pienemmän keon alkiot yksitellen suurempaan kekoon. keot tuhotaan yhdistämisen jälkeen ja funktio palauttaa uuden yhdistetyn keon. sallii eri asteisten kekojen yhdistämisen, haarautumisaste uuteen kekoon valitaan suuremman keon mukaan.
public static Dkeko mergeBottomUp(Dkeko t1,Dkeko t2)	Jos keot ovat yhtäsuuria bottomupmerge on suorituskyvyltää suositeltavampi. keot liitetään silloin yhteen alhaalta ylöspäin tasoittain kokoamalla.

1.1.4 class Kekoalkio

Dkeko luokan apuluokka. Kekoalkio luokan oliot muodostavat kaksisuuntaisen linkitetyn listan.

Kekoalkio luokka sisältää setterin ja getterin Solmu luokan olioon. Kekoalkio luokan jäsenmuuttuja keyn avulla lasketaan minimikeon vanhempi ja lapset. Vanhemman key arvo kerrotaan haarautumisasteella ja siihen lisätään lapsen järjestysnumero, alkaen 1 :stä. Täten saadaan selville lapsen key arvo ja löydetään lapsi linkitetystä listasta. Vanhempi löydetään vähentämällä lapsen key arvosta yksi ja jakamalla tulos haarautumisasteella.Luokan tarkempi kuvaus löytyy JavaDoc dokumentista.

1.1.5 class Binomikeko

Binomikeko tietorakenteena toteutettu minimikeko. Luokka toteuttaa perusoperaatiot binomikeko tietorakenteesta siten kuin ne on yleisesti kirjallisuudessa määritelty. Perusoperaatiot ovat kuvattu alla, mutta luokka sisältää myös tukifunktioita, jotka on kuvattu JavaDocissa. Binomikeko on toteutettu yksisuuntaisena linkitettynä listana,jossa alkioina on Binomipuu luokan oliota, jotka muodostavat linkitetynlistan rakenteen ja sisältävät Solmu oliot. Binomikeko muodostuu pienemmästä suurempaan järjestetystä juurilistasta,johon eriasteiset binomipuut on linkitetty. Järjestys perustuu binomipuu juuren lapsien lukumäärään, jota kutsutaan tässä asteeksi.Vain yksi kutakin astetta sallitaan juurilistassa, muutoin se aiheuttaa samanasteisten binomipuiden merge operaation. Binomikekoon osoitetaan juurilistan minimialkiolla. Siis alkiolla jonka asteluku on pienin.Juurilistasta löytyy minimikeon pienimmät alkiot kekoehdon mukaisesti. Kustakin minimikeon solmusta on linkki vanhempaan, lapseen ja oikeanpuoleiseen sisarukseen.

public Binomikeko()	Konstruktori.
public int decreaseKey(Binomipuu binomipuu , Solmu salkio , int value)	Funktio pienentää keossa olevan alkion key arvoa, jos parametrina annettu value arvo on pienempi kuin keossa olemassaoleva arvo.Kekoehdon rikkoutuessa suoritetaan ylöspäin korjaus.Palauttaa 0, jos asetus onnistuis, muutoin palauttaa -1.
public Solmu deleteMin()	Funktio poistaa minimikeosta keon pienimmän alkion ja palauttaa sen.Keosta poistetun alkion lapset muodostavat uuden Binomikeon ja se yhdistetään olemassa olevaan kekoon lomitus operaatiolla, jossa kahden keon juurilistat lomitetaan ja senjälkeen juurilista käydään läpi ja yhdistetän samanasteiset binomipuut.
public Solmu findMin()	Funktio palauttaa minimikeon pienimmän alkion eli Solmu olion, jonka value arvo on keon pienin. Solmu säilyy keossa. Keon jäsenmuuttuja juurilista min osoittaa juurilistan alkuun ja keon pienin löytyy juurilista läpikäymällä.
public void insert(Solmu x)	Funktio lisää Solmu luokan olion minimikekoon, olion omistajuus siirtyy.Uudesta minimikeko alkiosta muodostetaan uusi binomikeko, joka lomitetan ja yhdistetään olemassaolevan binomikeon kanssa. Tarkistetaan myös juurilistaminimin mahdollinen päivitystarve.
public static Binomikeko makeHeap()	Luo uuden tyhjän binomikeon.Juurilistanminimi saa arvon null.
public static Binomikeko merge(Binomikeko t1, Binomikeko t2)	Yhdistää kaksi kekoa toisiinsa, luoden uuden keon ja tuhoten keon t1 ja t2.Lomittaa juurilistat ja yhdistää saman degreen juurilistan alkiot toisiinsa.

1.1.6 class Binomipuu

Binomikeko luokan apuluokka. Binomipuu luokan oliot muodostavat yksisuuntaisen linkitetyn listan. Binomipuu sisältää linkin vanhempaan, lapseen, oikeanpuoleiseen sisarukseen, Solmu luokan olion sekä asteen, joka kuvaa alkion lapsien lukumäärää.Tarkempi kuvaus luokan metodeista on kuvattu javaDoc dokumentissa.

1.1.7 class Fibonaccikeko

Fibonaccikeko tietorakenteena toteutettu minimikeko. Luokka toteuttaa perusoperaatiot fibonaccikeko tietorakenteesta siten kuin ne on yleisesti kirjallisuudessa määritelty. Perusoperaatiot ovat kuvattu alla, mutta luokka sisältää myös tukifunktioita, jotka on kuvattu JavaDocissa. Fibonaccikeko on toteutettu kaksisuuntaisena linkitettynä rengaslistana,jossa alkioina on Fibonaccipuu luokan oliota, jotka muodostavat linkitetynlistan rakenteen ja sisältävät Solmu oliot. Fibonaccikeko muodostuu järjestymättömästä juurilistasta,johon eriasteiset binomipuut on linkitetty. Vain yksi kutakin astetta sallitaan juurilistassa, muutoin se aiheuttaa samanasteisten binomipuiden merge operaation. Fibonaccikekoon osoitetaan juurilistan minimialkiolla. Siis alkiolla jonka key arvo (Solmun luokan olion value arvo) on pienin.Juurilistasta löytyy minimikeon pienimmät alkiot kekoehdon mukaisesti. Kustakin minimikeon solmusta on linkki vanhempaan, lapseen, vasemman-ja oikeanpuoleiseen sisarukseen.Binomikeosta poiketen, Fibonaccikeon alkiosta talletetaan myös boolean tyyppinen markedInfo.

public Fibonaccikeko()	Konstruktori.
public int decreaseKey(Fibonaccipuu puu , int value)	Funktio pienentää keossa olevan alkion key arvoa, jos parametrina annettu value arvo on pienempi kuin keossa olemassaoleva arvo.Pienettävä keon alkio nostetaan juureen ja jos pienettävän vanhempi oli suurempi ,se merkitään boolean arvolla true markedInfo jäsenmuuttujaan. Jos arvo on jo valmiiksi true, myös vanhempi nostetaan juurilistaan ja tarkistetaan sen vanhempi ja näin toimitaan rekursiivisesti.Palauttaa 0, jos asetus onnistuis, muutoin palauttaa -1.
public Solmu deleteMin()	Funktio poistaa minimikeosta keon pienimmän alkion ja palauttaa sen.Keosta poistetun alkion lapset muodostavat uuden Binomikeon ja se yhdistetään olemassa olevaan kekoon limitys operaatiolla, jossa kahden keon juurilistat yhdistetään tarkistamatta samanarvoisia, vain minimialkio tarkistetaan mahdollisen päivityksen vuoksi.
public Solmu findMin()	Funktio palauttaa minimikeon pienimmän alkion eli Solmu olion, jonka value arvo on keon pienin. Solmu säilyy keossa. Keon jäsenmuuttuja juurilista min osoittaa suoraan keon minimialkioon.
public void insert(Solmu x)	Funktio lisää Solmu luokan olion minimikekoon, olion omistajuus siirtyy.Uusi minimikeko alkio vain

	limitetään jo olemassa olevaan minimikekoon. Tarkistetaan myös juurilistaminimin mahdollinen päivitystarve.
public static Fibonaccikeko makeHeap()	Luo uuden tyhjän fibonaccikeon.Juurilistan minimi saa arvon null.
public static Fibonaccikeko merge(Fibonaccikeko t1, Fibonaccikeko t2)	Yhdistää kaksi kekoa toisiinsa, luoden uuden keon ja tuhoten keon t1 ja t2.Limittää juurilistat ja tarkistetaan mahdollinen juurilistan minimin päivitystarve.Samanalkioisia binomipuita juurilistassa EI siis yhdistetä.

1.1.8 class Fibonaccipuu

Fibonaccikeko luokan apuluokka. Fibonaccipuu luokan oliot muodostavat kaksisuuntaisen linkitetyn rengaslistan. Fibonaccipuu sisältää linkin vanhempaan, lapseen, vasemman- ja oikeanpuoleiseen sisarukseen, Solmu luokan olion, merkitty tiedon sekä asteen, joka kuvaa alkion lapsien lukumäärää.Tarkempi kuvaus luokan metodeista on kuvattu javaDoc dokumentissa.

1.2 Saavutetut aika-ja tilavaativuudet

Dkeko

funktio	**Aikavaativuus runko**	**aliohjelma kutsu**	**Funktion AV:**	**Tilavaativuus**
makeHeap	O(1)		**O(1)**	**O(1)**
findMin	O(1)		**O(1)**	**O(1)**
insert	O(1)	1*decreaseKey()	**O(log n)**	**O(1)**
deleteMin	O(1)	h*minHeapify O(log n)	**O(log n)**	**O(1)**
minHeapify (rekursiivinen)	0(1)	d*getChild	O(log n)	
		vaihdaPaikkaa()		
		h*minHeapify() O(log n)		
vaihdaPaikkaa()	O(1)		O(1)	
getChild	O(1)	findSolmu()	O(n)	O(1)
findSolmu()	O(n)		O(n)	
decraseKey	O(1)	1*vaihdaJarjestys()	**O(log n)**	**O(1)**
vaihdajarjestys (rekursiivinen)		1*countParent()	O(log n)	
		1*findSolmu	O(n)	
		vaihdapaikkaa()	O(1)	
		h*vaihdajarjestys O(log n)		
countParent()	O(1)		O(1)	
merge	O(1)	n*deleteMin()	**O(log n)**	**O(1)**
		n*insert()		
mergeBottomUp		n*asetaKeyt() (rekursiivinen)	**O(log n)**	
asetakeyt	O(1)	n*minHeapify()		
findKekoalkio	O(n)		O(n)	
getHeapSize	O(1)		O(1)	
getTail	O(1)		O(1)	
getMin	O(1)		O(1)	
getAste	O(1)		O(1)	

Kekoalkio

getLeft	O(1)			O(1)
getRight	O(1)			O(1)
setLeft	O(1)			O(1)
setRight	O(1)			O(1)
getValue	O(1)			O(1)
setValue	O(1)			O(1)
getKey	O(1)			O(1)
setKey	O(1)			O(1)

Binomikeko

funktio	**Aikavaativuus runko**	**aliohjelma kutsu**	**Funktion AV:**	**Tilavaativuus**
makeHeap	O(1)		**O(1)**	**O(1)**
findMin	O(log n)		**O(log n)**	
insert		createNewBinomipuu	**O(log n)**	
		makeHeap		
		merge		
createNewBinomipuu	O(1)			O(1)
link	O(1)			
lomitaJuuriListat	O(log n)	makeHeap		O(1)
deleteMin	O(1)	makeHeap	**O(log n)**	
		insertBinomipuu		
		getChild		
		merge		
insertBinomipuu	O(log n)	insertBinomipuu rekursiivinen *lasten lkm		
findAndRemoveMin	O(n)	removeMin		
removeMin	O(1)			
decreaseKey	O(log n)	vaihdaPaikkaa	**O(log n)**	
vaihdaPaikkaa	O(1)			
merge	O(1)	lomitaJuuriListat	**O(log n)**	
		yhdistaJuuriLista		
yhdistaJuuriLista	O(1)			O(1)
getHeapSize	O(n)			
getJuuriListaMin	O(1)			
setJuuriListMin	O(1)			
findBinomipuu	O(n)			O(1)

Binomipuu

getChild	O(1)			O(1)
setChild	O(1)			O(1)

getSibling	O(1)			O(1)
setSibling	O(1)			O(1)
getValue	O(1)			O(1)
setValue	O(1)			O(1)
getParent	O(1)			O(1)
setParent	O(1)			O(1)
setDegree	O(1)			O(1)
getDegree	O(1)			O(1)

Fibonaccikeko

funktio	**Aikavaativuus runko**	**aliohjelma kutsu**	**Funktio AV:**	**Tilavaativuus**
makeHeap	O(1)		**O(1)**	
findMin	O(1)		**O(1)**	
insert	o(1)	createNewFibonaccipuu	**O(1)**	
		makeHeap		
		merge		
createNewFibonaccipuu	O(1)			
merge	O(1)	nullVersionMerge	**O(1)**	
		makeHeap		
		sulauta		
nullVersionMerge	O(1)			
getMin	O(1)			
setMin	O(1)			
deleteMin	O(1)	getMin	**O(n)**	
		setMin		
		removelistasta		
		sulauta		
		puhdista		
removelistasta	O(1)			
sulauta	O(1)			
puhdista	O(1)	n*yhdistaSamanasteiset		O(n)
		getMin		
yhdistaSamanasteiset	O(1)?	removelistasta		O(1)
		link		
link	O(1)	sulauta		
vaihdaArvot	O(1)			
decreaseKey	O(1)	findMin	**O(log n)**	O(1)
		setMin		
		korota		
korota		removelistasta		
		sulauta		

		korota		
findFibonaccipuu	O(n)	rekursiivinen		O(1)

Fibonaccipuu

getChild	O(1)			O(1)
setChild	O(1)			O(1)
hasSibling	O(1)			O(1)
isSameSibling	O(1)			O(1)
getSiblingL	O(1)			O(1)
getSiblingR	O(1)			O(1)
setSiblingL	O(1)			O(1)
setSiblingR	O(1)			O(1)
getValue	O(1)			O(1)
setValue	O(1)			O(1)
getParent	O(1)			O(1)
setParent	O(1)			O(1)
getMarkedInfo	O(1)			O(1)
setMarkedInfo	O(1)			O(1)
setDegree	O(1)			O(1)
getDegree	O(1)			O(1)

1.3 Suorituskyky-ja O-analyysivertailu

Taulukossa esitetty määrittelydokumentissa ilmoitetut kirjallisuudessa määritetyt aikavaativuudet.

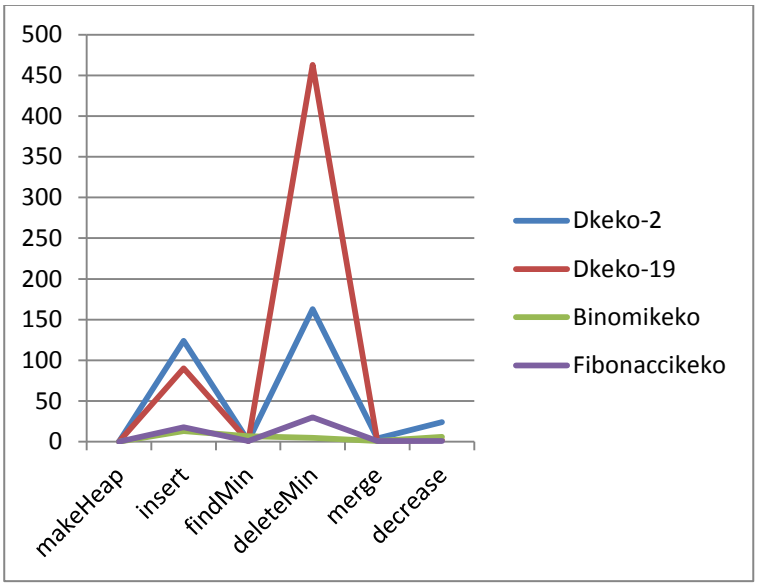
Keko	**makeHeap**	**findMin**	**insert**	**deleteMin**	**decreasekey**	**merge**
D-keko	O(1)	O(1)	O(log n)	O(log n)	O(log n)	O(M log n)
Binomikeko	O(1)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
Fibonaccikeko	O(1)	O(1)	O(1)	O(n)	O(n)	

Taulukossa alla esitetty toteutuksen aikavaativuudet

Keko	**makeHeap**	**findMin**	**insert**	**deleteMin**	**decreasekey**	**merge**
D-keko	O(1)	O(1)	O(log n)	O(log n)	O(log n)	O(M log n)
Binomikeko	O(1)	O(log n)	O(log n)	O(log n)	O(log n)	O(log n)
Fibonaccikeko	O(1)	O(1)	O(1)	O(n)	O(log n)	

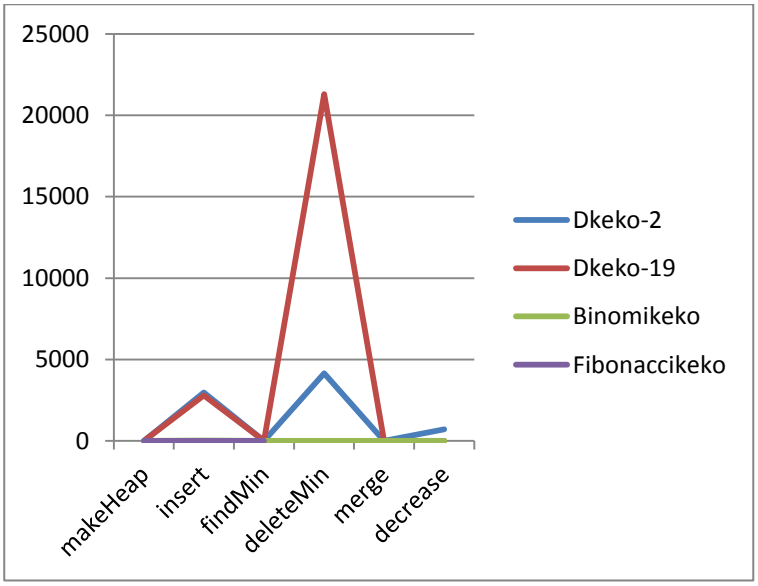
Syöte lukumäärä 6002 kpl /aika millisekuntteja

	makeHeap	insert	findMin	deleteMin	merge	decrease
Dkeko-2	0	124	1	163	4	24
Dkeko-19	0	90	1	463	1	2
Binomikeko	0	13	7	5	1	6
Fibonaccikeko	0	18	1	30	1	1



Syöte lukumäärä 30002 kpl(aika millisekuntteja)

	makeHeap	insert	findMin	deleteMin	merge	decrease
Dkeko-2	0	2971	1	4147	5	723
Dkeko-19	0	2805	1	21295	3	2
Binomikeko	0	19	3	7	0	4
Fibonaccikeko	0	10	1	NA	NA	NA



2 OHJELMAN MAHDOLLISET PARANNUSEHDOTUKSET

Sovelluksen refactorointi suorituksen optimoimiseksi tulisi aloittaa Fibonacciluokasta. Luokan insert metodi luo turhaan uuden minimikeon ja liittää sen mergellä olemassaolevaan kekoon. Kyseisen toimenpiteen voisi tehdä vain nostamalla luotu Fibonaccipuu olio suoraan minimin tarkistuksen jälkeen juurilistaan. Myös apufunktioiden lyhentäminen ja selkeyttäminen tulisi tehdä koodin ylläpidon onnistumisen vuoksi.

Sovelluksen Kekoharjoitus luokan suorituskyky vertailu testit tulisi rakentaa siten, että keot voitaisiin esitttää interfacen kautta ja siten yksinkertaistaa kutsuja ja lyhentää suorituskykytesti tapauksia. Tämä jäi ensimmäisestä vaiheesta ajanpuutteen vuoksi tekemättä. Myös kekojen vertailu toimivana prioriteettijonona olisi vaatinut lisää aikaa. Eli sen totetuttaminen olisi tulevaisuuden parannusehdotuksista yksi päällimmäisistä.

Myös mahdollisen graafisen käyttöliittymän suunnittelu, jossa eri kekojen nopeus eri syötteillä voitaisiin esittää graafisina automaattisina taulukoina olisi mahdollinen parannusehdotus.

Fibonaccikeon kaksisuuntainen rengaslista monimutkaisti selkeästi koodia. Parannusehdotuksena olisi muunmuassa tunnussolmullisen kaksisuuntaisen rengaslistan teko. Nyt yhden, kahden ja useamman solmun listaa käsitellään turhaan hieman eri tavoin, Ylimääräisiä tarkastuksia linkkien oikeellisuuden tarkistamiseksi joudutaan tekemään.

Fibonaccikeon aputaulukko pudistus metodin yhteydessä tulisi myös korjata. Aputaulukon maximiarvon kanssa tulee satunnaisesti stack overload virhetilanteita.

3 LIITTEET

LIITE 1. Testausdokumentti

LIITE 2. JavaDokumentaatio

LIITE3. Kayttoohje

4 LÄHTEET

http://www.oracle.com/technetwork/java/codeconv-138413.html

http://www.cs.helsinki.fi/u/tapasane/keot.pdf

http://www2.it.lut.fi/kurssit/00-01/010534000/luennot/penttonen/osa8.html

http://trakla.cs.hut.fi/ebook/ebook-Keko.html

http://www.cs.helsinki.fi/courses/58131/2013/k/k/1

”Johdatus algoritmien suunnitteluun ja analysointiin” Martti Penttonen,Otatieto,ISBN 951-672-249-0

”Introduction to ALGORITHMS” Third edition, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, ISBN 978-0-262-03384-8