

TiRa labra - Toteutusdokumentti

Jarmo Isotalo

October 6, 2012

1 Toteutettavat algoritmit

Toteutettavat algoritmit muuttuivat kurssin aikana. Päädyin lopulta toteuttamaan seuraavat tietorakenteet; binääri-keon, kolmikeon ja d-keon.

2 Toteutuneet aika- ja tilavaativuudet (O-analyysi)

2.1 Aikavaatimus

Tarkastelen tässä vain muutaman eri tapauksen aikavaativuuksia:

1. Keon alustaminen - Create
Jokaisessa keossa, binääri-, kolmi-, ja d-keossa operaatio on kutakuinkin saman kestoinen:
 - (a) Aluksi alustetaan taulukko ja tallennetaan tietoon kunkin lapsien määrä. Binääri-keossa lapsia on kaksi, kolmikeossa kolme ja d-keossa d kappaletta. $O(1)$
 - (b) Koska Create tehdään tyhjälle keolle, on operaatio vakioaikainen. Tässä asetetaan taulun ensimmäiseen indeksiin parametrina saatu arvo. $O(1)$
2. Kekoon lisääminen - Insert
Jokaisessa keossa, binääri-, kolmi-, ja d-keossa operaatio on kutakuinkin saman kestoinen:
 - (a) Aluksi parametrina saatu elementti lisätään keon viimeiseen indeksiin. $O(1)$
 - (b) Sitten indeksille suoritetaan *heapify-up*, joka siirtää elementtiä ylöspäin, kunnes keko noudattaa taas kekoehto. Tässä oletetaan, että keko noudatti kekoehto ennen elementin lisäämistä.

Tätä tapahtuu keon korkeuden verran. Eli insertin aikavaativuus on toteutuksessani $O(\log n)$

3. Keosta poistaminen - Delete

Jokaisessa keossa, binääri-, kolmi-, ja d-keossa operaatio on kutakuinkin saman kestoinen:

- (a) Elementtiä keosta poistettaessa poistetaan elementti keon taulukon indeksistä 0. $O(1)$
- (b) Sen jälkeen siirretään keossa viimeisenä oleva elementti kekotaulukon indeksiin nolla. $O(1)$
- (c) Sitten kutsutaan *heapify_down* äsekettäin indeksiin nolla siirretylle, kunnes kekoehto toteutuu. $O(\log n)$. Lisäksi *heapify_down* tarkastaa onko elementillä suurempia lapsia. Tämän aikavaativuus on

Kekojen toteutuksen vuoksi O notaation ajat ovat samankaltaisia, mutta todellisuudessa lasten määrän lisääminen nopeuttaa keon toimintaa. Kunnolliset BenchMarkit tulossa TODO

	Binary Heap	Three Heap	D-ary Heap
Create	$O(1)$	$O(1)$	$O(1)$
Insert	$O(\log n)$	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$	$O(\log n)$

2.2 Tilavaativuus

Tarkastelen tässä vain niiden metodien tilavaativuuksia, joiden aikavaativuudet yllä1:

1. Keon alustaminen - Create

Jokaisessa keossa, binääri-, kolmi-, ja d-keossa operaatio on kutakuinkin saman kestoinen:

- (a) Aluksi alustetaan taulukko ja tallennetaan tietoon kunkin lapsien määrä. Binääri-keossa lapsia on kaksi, kolmikeossa kolme ja d-keossa d kappaletta. $O(1)$
- (b) Koska Create tehdään tyhjälle keolle, on operaatio vakiotilainen. Tässä asetetaan taulun ensimmäiseen indeksiin parametrina saatu arvo. $O(1)$. Apumuuttujia ei tarvita.

2. Kekoon lisääminen - Insert

Jokaisessa keossa, binääri-, kolmi-, ja d-keossa operaatio on kutakuinkin saman kestoinen:

- (a) Aluksi parametrina saatu elementti lisätään keon viimeiseen indeksiin. $O(1)$
- (b) Sitten indeksille suoritetaan *heapify_up*, joka siirtää elementtiä ylöspäin, kunnes keko noudattaa taas kekoehto. Tässä oletetaan, että keko noudatti kekoehto ennen elementin lisäämistä. *heapify_up* -metodia kutsutaan siis rekursiivisesti. Tätä tapahtuu keon korkeuden verran. Insertin tilavaatimus on rekursiopinin kokoinen eli $O(\log n)$

3. Keosta poistaminen - Delete

Jokaisessa keossa, binääri-, kolmi-, ja d-keossa operaatio on kutakuinkin saman kestoinen:

- (a) Elementtiä keosta poistettaessa poistetaan elementti keon taulukon indeksistä 0. $O(1)$
- (b) Sen jälkeen siirretään keossa viimeisenä oleva elementti kekotaulukon indeksiin nolla. $O(1)$
- (c) Sitten kutsutaan *heapify_down* äsekettäin indeksiin nolla siirretylle, kunnes kekoehto toteutuu. *heapify_down* -metodia kutsutaan rekursiivisesti eli sen tilavaatimus on rekursiopinin kokoinen $O(\log n)$ Lisäksi *heapify_down* tarkastaa onko elementillä suurempia lapsia. Tämän tilavaativuus on $O(d)$, jossa d on lasten määrä eli vakio. tilavaatius on

	Binary Heap	Three Heap	D-ary Heap
Create	$O(1)$	$O(1)$	$O(1)$
Insert	$O(\log n)$	$O(\log n)$	$O(\log n)$
Delete	$O(\log n)$	$O(\log n)$	$O(\log n)$

3 Tiedostojen siainnit

Tiedostojen polut on annettu suhteellisesti git repositorion juuresta katsotuna.

- Dokumentaatio
 - docs/
 - * Suunnitteludokumentti
 - * Toteutusdokumentti
 - * Käyttöohje
 - * Testausdokumentti

- Lähdekoodi
 - *heaps/Heap/src/*
 - * Binäärikeko - `binary_heap.rb`
 - * 3-keko - `three_heap.rb`
 - * D-keko - `d_heap.rb`
 - * Pino - `stack.rb`
- Testit
 - *heaps/Heap/spec/*
 - * Binäärikeon testit - `binary_heap_spec.rb`
 - * 3-keon testit - `three_heap_spec.rb`
 - * D-keon testit - `d_heap_spec.rb`
 - * Pinon testit - `stack_spec.rb`
- Testien kattavuus
 - *heaps/Heap/src/doc*
 - * `index.html` - Kaunis html pohjainen esitys koodin kattavuudesta

4 Lähteet

- Keot yleisesti
 - [https://en.wikipedia.org/wiki/Heap_\(data_structure\)](https://en.wikipedia.org/wiki/Heap_(data_structure))
 - <http://www.cs.helsinki.fi/u/tapasane/keot.pdf>
 - <http://www.cs.helsinki.fi/u/floreen/tira2012/tira.pdf>
- Binäärikeko
 - http://en.wikipedia.org/wiki/Binary_heap
- 3-keko
 - <http://www.cs.helsinki.fi/u/floreen/tira2012/teht07.pdf>
- D-keko
 - http://en.wikipedia.org/wiki/D-ary_heap