

Prioriteettijonot

Ari Korhonen

7. Prioriteettijonot (priority queues)

[7.1 Abstrakti tietotyyppi prioriteettijono](#)

[7.2 Prioriteettijonon toteutuksia: keko \(heap\)](#)

[7.3 Prioriteettijono sovelluksia: kekojärjestäminen \(heap sort\)](#)

2/16/05

2

7.1 Prioriteettijono

- Abstrakti tietotyyppi, jonka toteutuksessa alkioihin liittyy *prioriteetti* (etuoikeusarvo)

Huom! toteutus ei ole välttämättä lineaarinen tietorakenne (prioriteettijono ei ole sama asia kuin jono)

- Prioriteettijonosta poistetaan aina pienin (suurin) alkio. Vertaa
 - Pino (poista uusin)
 - Jono (poista vanhin)

Prioriteettijonoissa esiintyviä operaatioita:

void **Add**(PriorityElement e) - lisää uusi alkio e prioriteetilla P
PriorityElement **DeleteMin**() - poista pienin (suurin) alkio
void **Replace**(PriorityElement e) - korvaa pienin alkio
void **Change**(PriorityElement e) - vaihda e:n prioriteetiksi P
PriorityElement **Delete**(PriorityElement e) - poista alkio e
PriorityQueue **Merge**(PriorityQueue Q1, Q2) - yhdistä kaksi jonoa

2/16/05

3

- Prioriteettijonoille on monia erilaisia implementaatioita. Niiden olennainen ero on siinä, miten tehokkaasti ne toteuttavat em. operaatioita.

Esimerkkejä:

1) Ei-järjestetty lista

- insert nopea, $O(1)$
- deletemin, replace, delete hitaita, $O(N)$

2) Järjestetty lista

- insert hidas, $O(N)$
- deletemin nopea, $O(1)$
- replace, delete hitaita, $O(N)$

2/16/05

4

3) Binäärinen hakupuu

- kaikki operaatiot $\log N$ keskimäärin (pahin tapaus $O(N)$)
- deletemin: poistetaan aina pienin eli vasemmanpuolisin alkio

4) Tasapainotettu puu

- pahin tapaus $O(\log N)$

- Hakupuut tukevat myös monia muita operaatioita ja ovat useimmiten turhan monimutkaisia prioriteettijonojen toteutukseen

5) Keko (ks. 7.2)

2/16/05

5

SOVELLUTUKSIA

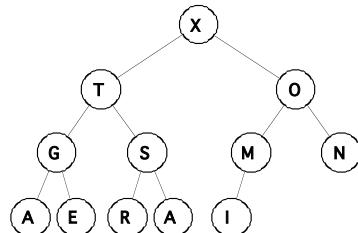
- Mikä tahansa prioriteettijono vastaa jotain järjestämisalgoritmia:
 - Lisää N alkioita ja poista ne yksitellen
 - ✓ Ei-järjestetty lista \Leftrightarrow selection sort
 - ✓ Järjestetty lista \Leftrightarrow insertion sort
- Printterijono
 - käyttäjän mukaan
 - tiedoston koon mukaan
- Simulaatiot
- Huffman encoding [Huffman 1952]
- Käyttöjärjestelmät
 - prosessien iän mukaan

2/16/05

6

7.2 Keko (heap)

- Tärkeä prioriteettijonototeutus. Kaikki operaatiot (paitsi Join-operaatio) ajassa $O(\log N)$
- Keko on binääripuu, jossa avain on pienempi (suurempi), kuin sen lapset. Tällöin sanotaan, että puu täyttää *kekoehdon*
- Esimerkin kekoehdo: isä suurempi kuin poika



2/16/05

7

- Keko voidaan esittää taulukkona, koska keko on täydellinen binääripuu

```

1 2 3 4 5 6 7 8 9 10 11 12
X T O G S M N A E R A I

```

```

isä :      j / 2
pojat :    2j ja 2j + 1

```

Keko-algoritmien yleisperiaate:

- Ensin yksinkertainen rakenteellinen muutos
- Sitten keon modifiointi siten, että kekoehdo palautetaan voimaan

2/16/05

8

Insert

- **Lisää alkio taulukon loppuun ensimmäiseen tyhjään paikkaan**
- **Korjaa kekoehto nostamalla alkiota ylemmäs vaihtamalla aina pareittain isän kanssa (upheap) tai luomalla tyhjä solmu, siirtämällä sitä ylöspäin ja asettamalla uusi alkio lopulta siihen**

DeleteMin

- **Juuri poistetaan**
- **Asetetaan viimeinen elementti juuren tilalle ja annetaan juurena olevan elementin 'valua' oikealle paikalleen (downheap) tai valutetaan tyhjää paikkaa alemmas ja siirretään viimeinen elementti lopulta siihen**
- **Vaihdetaan isä aina pienemmän (riippuen kekoehdosta) pojan kanssa**

Esimerkki prioriteettijonon käytöstä, valikointiongelmaksi: Etsi taulukosta k:nneksi pienin alkio!**Menetelmä A**

- **Luetaan N alkiota taulukkoon**
- **Järjestäminen $O(N^2)$ tai $O(N \log N)$**
- **Otetaan k:nnes alkio**

Menetelmä B

- **Luetaan $k \ll N$ alkiota taulukkoon**
- **Järjestetään kasvavaan järjestykseen $O(k^2)$**
- **Nyt k:nnes on suurin**
- **Luetaan loput $N-k$ alkiota yksi kerrallaan**
 - **Jos luettu alkio pienempi kuin k:nnes**
=>
✓ **Sijoitetaan uusi alkio paikalleen**
✓ **k:nnes häviää**
✓ **$O((N - k) k)$**
- **Lopussa k:nneksi pienin on k:nnentena**
 - **Jos k on 'pieni' => $O(N k)$**
 - **Jos k on luokkaa $n/2$ => $O(N^2)$**

Menetelmä C

- **Sijoitetaan N alkiota kekoon**
 - $O(N \log N)$ pahin tapaus
 - $O(N)$ keskimäärin, sillä insert menee keskimäärin ajassa $O(1)$
 - Poistetaan k alkiota $O(k \log N)$
 - Viimeksi poistettu alkio on k:nneksi pienin
 - Kokonaisaika $O(N + k \log N)$

2/16/05

13

Menetelmä D

- **Sijoitetaan k alkiota kekoon $O(k)$**
 - **Kekoehto: isä suurempi kuin poika**
 - **Nyt suurin on kekon päällä**
- **Luetaan loput N-k yksi kerrallaan**
 - **Jos luettu alkio on pienempi kuin keon päällimmäinen**
=>
 - ✓ **Poistetaan päällimmäinen (suurin) $O(\log k)$**
 - ✓ **Laitetaan uusi alkio kekoon $O(\log k)$**

2/16/05

14

- **Lopussa k:nneksi pienin keon päällä**
- **Kokonaisaika $O(N + (N - k) \log k) = O(N \log k)$**

Huom!

- **k:nneksi pienin alkio on $(N - k + 1)$:nneksi suurin alkio**

2/16/05

15

Esimerkki 2, tapahtumasimulointi:

- **Esim. Jonotus rautateiden lipunmyynnissä**
- **Uusia asiakkaita generoituu vaikkapa eksponentiaalijakauman mukaan**
- **Asiakkaan palveluaika jakautuu jonkin toisen jakauman mukaan**
- **Simuloinnin perusteella määrätään palvelevien kassojen lukumäärä**
- **Kullakin hetkellä tiedetään joukko tulevia tapahtumia ja niiden tapahtuma-ajat**
- **Tulevat tapahtumat kannattaa tallettaa kekoon, jossa prioriteetti arvona on aika**

2/16/05

16

7.3 Kekojärjestäminen (heap sort)

- Lisätään kaikki alkiot kekoon ja poistetaan ne yksitellen
- Poistetut alkiot voidaan sijoittaa samaan tilaan

```
for (k = 1; k <= M; k++)  
    insert(taulu[k]);
```

```
for (k = M; k > 0; k--)  
    taulu[k] = deletemin(taulu);
```

- Menetelmä on riippumaton siitä, miten prioriteettijono on toteutettu. Jos toteutus on keko, saadaan kekojärjestäminen.
- Kekojärjestämisessä tehokkuusluku on aina korkeintaan $N \log N$. Suoritettavia käskyjä / elementti kuitenkin enemmän kuin Quicksortissa keskimäärin