
Tietorakenteet ja algoritmit

Lineaariset perustietorakenteet

Ari Korhonen

4. LINEAARISET PERUSTIETORAKENTEET

[4.1 Johdanto](#)

[4.2 Taulukko](#)

[4.3 Linkitetty lista \(linked list\)](#)

[4.4 Pino \(stack\)](#)

[4.5 Jono \(queue\)](#)

[4.6 Pakka \(deque\)](#)

4.1 Johdanto

- Tietorakenne on lineaarinen, jos kaikki sen alkiot on ryhmitetty peräkkäin.
- Tällaista rakennetta voidaan käyttää esim.
 - yksinkertaisena hakurakenteena,
 - tilapäisenä tallennusrakenteena.

4.2 Taulukko

- Hyvin yksinkertainen lineaarinen rakenne.
- Perusrakenne valmiina useimmissa korkean tason ohjelmointikielissä.
 - => Helppo ohjelmoida
- Alkiot indeksoidaan ensimmäisestä viimeiseen.
- Mihin tahansa alkioon voidaan heti viitata.
- Taulukon alkioden väliin ei voi lisätä uutta alkiota eikä sieltä voi poistaa alkiota.
 - => Rakenne on erittäin jäykkä, jos tällaisia toimintoja (päivityksiä) tarvitaan.

4.3 Linkitetty lista (linked list)

- Taulukkoa paljon joustavampi rakenne monessa tilanteessa.
- Lista koostuu yksittäisistä alkioista, joihin kuhunkin on talletettu dataa ja *osoitin (referenssi)* seuraavaan alkioon.
- Listan kokoa ei ole ennalta rajoitettu, vaan listaan voidaan lisätä ja siitä voidaan poistaa alkioita ajonaikaisesti.
=> Listan kokoa rajoittaa vain käytettävissä oleva muisti.
- Alkioita voi lisätä minne tahansa.
- Alkioita voi poistaa mistä tahansa.
- Alkioiden järjestystä voi vaihtaa.
- Listan alkiot voivat olla samaa tyyppiä tai lista voi olla *geneerinen*.

3.2.2004

5

LINKITETYN LISTAN TÄRKEIN RAJOITUS:

- Alkioita voi käydä läpi vain yksitellen, ts. alkioihin ei voi viitata suoraan (vrt. taulukko)

Linkitetyn listan tärkeimmät operaatiot ovat:

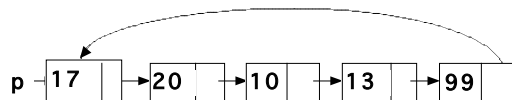
- TULOSTUS
- HAKU
- LISÄYS
- POISTO
- PÄIVITYS
- TYHJÄN LISTAN LUOMINEN
- N:NNEN ALKION PALAUTTAMINEN
- ONKO LISTA TYHJÄ?

3.2.2004

...

6

- Eri operaatioitten vaatima aika:
 - TULOSTUS N
 - HAKU N/2 (epäonnistunut: N)
 - LISÄYS 1
 - POISTO N/2
- Lista voidaan pitää järjestyksessä, jolloin (epäonnistunut) haku on nopeampi
 - Tiedetään, ettei enää kannata etsiä, kun oikea kohta on jo ohitettu
 - Lisäys on vastaavasti hitaampi (ei voidakaan lisätä aina listan alkuun)
- Listasta voidaan muodostaa *rengas*



3.2.2004

K2.6

7

4.4 Pino (stack)

- Toimii samaan tapaan kuin pöydällä oleva korttipakka
- Pino on abstrakti tietotyyppi, jolle on määritelty seuraavat operaatiot:
 - Push(x) - lisää pinon päälle alkion x
 - Pop() - palauttaa ja poistaa pinon päällimmäisen alkion
 - Top() - palauttaa pinon päällimmäisen alkion (poistamatta sitä)
 - IsEmpty() - palauttaa tiedon siitä, onko pino tyhjä

3.2.2004

8

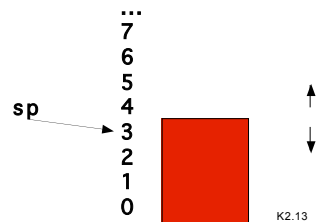
Jonokuri LIFO (Last In - First Out)

- Kaikki pino-operaatiot voidaan toteuttaa vakioajassa (ei riipu pinon koosta)
- Pino voidaan implementoida taulukon avulla tai linkitettyinä listana

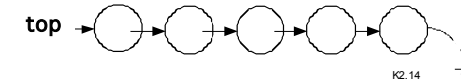
Toteutus taulukkona:

- apuna *pino-osoitin*, joka osoittaa päällimmäiseen alkioon

- isEmpty() \Leftrightarrow (sp == -1)
- Taulukkototeutuksessa pinolle varattu tila voi loppua



- Toteutus listana:



- push(): liitä uusi alkio listan alkuun
- pop(): poista ja palauta alkio listan alusta
- isEmpty(): tosi, jos lista on tyhjä (top == null)

Pinon sovelluksia

- Ohjelmointikielessä lausesulkujen tarkastus: kullekin alkusululle löytyy oikeantyyppinen loppusulku
 - Kielessä voi esiintyä useita eri sulkutyyppejä, esim. C-kielessä (,); [,]; {,}
 - Sulkuvirheet (esim. "{()}") voivat usein sotkea kääntäjän toiminnan pahoin
- Ajonaikainen pino (aliohjelmakutsut):
 - Kutsuvan ohjelman tila (=rekisterit) ja kutsun paikka (=program counter) pinoon
 - Aliohjelmasta palattaessa voidaan palauttaa alkuperäinen tila
- Syötevirran kääntäminen päinvastaiseen järjestykseen

Yksinkertainen lausesulkujen tarkastusperiaate

- Luetaan syötettä merkki kerrallaan
- Tutkitaan vain sulkumerkkejä
- Alkusulku asetetaan pinoon (push)
- Kohdattaessa loppusulku otetaan pinosta (pop) päällimmäinen sulku
 - ✓ Jos pino on tyhjä (isEmpty) tai sulku on väärän tyyppinen \Rightarrow virhe
- Algoritmin päättyessä pinon tulee olla tyhjä

Lausekkeen arvon laskeminen

- Miten evaluoida lausekkeen arvo?
- **Infix-notaatio:**
 $5 * ((9 + 8) * 4 * 6 + 7)$
- Miten tallentaa välitulokset?
- Mikä on toimintajärjestys, kun lauseke luetaan ohjelmalle merkki merkiltä?

Periaate:

- 1) Muunnetaan lauseke pinon avulla postfix-muotoon:

$5 \ 9 \ 8 \ + \ 4 \ * \ 6 \ * \ 7 \ + \ *$

- 2) Evaluoidaan postfix-lauseke pinon avulla

INFIX-POSTFIX-muunnos pinon avulla

- Luetaan Infix-lauseketta merkki kerrallaan
- operandit [0-9]*
⇒ tulostetaan suoraan
- (
⇒ asetetaan pinoon (PUSH)
- operaattorit *,+
⇒ otetaan pinosta (POP) ja tulostetaan operaattoreita, joilla on sama tai suurempi prioriteetti, sen jälkeen asetetaan viimeksi luettu operaattori pinoon. Sulkumerkkiä ')' ei kuitenkaan lueta
-)
⇒ otetaan pinosta ja tulostetaan operaattoreita, kunnes löytyy alkusulku, joka myös luetaan, mutta ei tulosteta
- Kun syöte loppuu, otetaan pinosta ja tulostetaan loput operaattorit

- **Esim. Muunnetaan lauseke $5 * ((9 + 8) * 4 * 6 + 7)$ muotoon $598+4*6*7+*$**

<u>Syöte</u>	<u>Pino</u>	<u>Tuloste</u>
5		5
*	*	
(*(
(*((
9		9
+	*((+	
8		8
)	*(+
*	*(*	
4		4
*	*(*	*
6		6
+	*(+	*
7		7
)	*	+
tyhjä		*

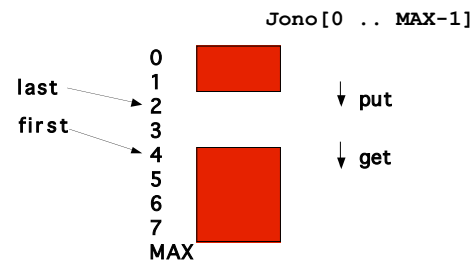
- **POSTFIX-lausekkeen arvo voidaan laskea pinon avulla**
- **Esim. lauseke 598+4*6*7+***

<u>Input</u>	<u>Pino</u>
5	5
9	5 9
8	5 9 8
+	5 17
4	5 17 4
*	5 68
6	5 68 6
*	5 408
7	5 408 7
+	5 415
*	2075
empty	

4.5 JONO (queue)

- **Jono on abstrakti tietotyyppi, jolle on määritelty seuraavat operaatiot:**
 - **Put(x) tai Enqueue(x)** - lisää jonon loppuun alkion x
 - **Get() tai Dequeue()** - palauttaa (ja poistaa) jonon ensimmäisen alkion
 - **First()** - palauttaa jonon ensimmäisen alkion
 - **IsEmpty()** - kertoo, onko jono tyhjä
 - **Jonokuri FIFO (First In First Out)**
 - **Jono voidaan toteuttaa eri tavoin (taulukon avulla tai linkitettyinä listana). Toteutetaan usein *sirkulaarisena taulukkona* (indeksit lasketaan modulo N).**

- **Jonon toteutus sirkulaarisena taulukkona**



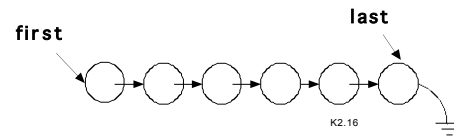
- **first osoittaa jonon alkuun**
- **last osoittaa 1. vapaaseen paikkaan**
- **Jonossa yksi tyhjä paikka, jota ei käytetä**

- **Put(x):**
`Jono[last] = x;
last = (last+1) % MAX;
if (last == first) error;
{Jono ei saa mennä täyteen!}`
- **Get():**
`tmp = Jono[first];
first = (first+1) % MAX;
return tmp;`
- **First():**
`return Jono[first];`
- **IsEmpty():**
`(last == first);`

▪ Jonon toteutus listana

```
struct node {
    datatype data;
    struct node *next;
};

typedef struct node * Queue;
```



3.2.2004

21

- **Put(x):**

```
temp = (Queue) malloc(sizeof(struct node));
temp->data = x;
temp->next = NULL;
last->next = temp;
last = temp;
```
- **Get():**

```
temp = first;
first = first->next;
return (temp);
```
- **First():**

```
return (first);
```
- **IsEmpty():**

```
return (first == NULL);
```

3.2.2004

22

Jonon sovelluksia

- **Esim. Printterijono**
- **Puskuri**
 - tiedon välivarasto
 - tietoa tulee
 - tietoa lähtee
 - tiedon järjestys säilyy
 - Esim. tulostettava tiedosto menee puskurin kautta
- **Diskreetti simulointi**
 - Esim. liikenne, tietoliikenne, palvelut
- **Verkkoalgoritmit (esim. leveyssuuntainen haku eli BFS)**

3.2.2004

23

4.6 PAKKA (deque)

- Alkioita voidaan asettaa rakenteen molempiin päihin, ja myös ottaa pois
- Vrt. korttipakka kädessä
- Määrittely jätetään kotitehtäväksi

3.2.2004

24

Ensi kerraksi

- Määrittele abstrakti tietotyyppi pakka
- Kuinka toteuttaisit määrittelemäsi abstraktion?
- Arvioi linkitetyle listalle määriteltyjen operaatioiden suoritusaikaa suhteessa listan pituuteen
- Vertaa linkitettyä listaa taulukkoon em. operaatioiden suhteen
- Lue kirjasta algoritmianalyysiä ja erityisesti funktioiden kasvunopeuksia ja ”O-notaatiota” käsittelevät kohdat
- Ensi kerralla:
 - Luennon aiheina puurakenteet, vieruslista, vierusmatriisi sekä algoritmianalyysi