811312A Tietorakenteet ja algoritmit, 2018 - 2019, Harjoitus 1 Ratkaisu

Harjoituksen aiheena ovat pinot, jonot ja listat

Tehtävä 1.1 Olkoon L lista, jossa on 1000 alkiota. Täytä alla olevaan taulukkoon ensimmäisessä sarakkeessa esitettyjen toimenpiteiden vaatimien operaatioiden lukumäärä huonoimmassa tapauksessa. **Alla parametri k tarkoittaa etsittävää avainta ja x osoitinta lisättävään/poistettavaan listan alkioon**. Listan mahdolliset tyypit on lueteltu taulukon ensimmäisellä rivillä. Oletetaan, että kahteen suuntaan järjestetyssä listassa pidetään yllä listan pään lisäksi listan hännän osoitinta, mutta yhteen suuntaan järjestetyssä listassa ylläpidetään ainoastaan listan pään osoitinta. Perustele myös vastauksesi. Kukin seuraavista lasketaan yhdeksi operaatioksi: siirtyminen solmusta toiseen, avaimen vertailu ja linkin muuttaminen.

	Yhteen suuntaan linkitetty järjestämätön lista	Yhteen suuntaan linkitetty järjestetty lista	Kahteen suuntaan linkitetty järjestämätön lista	Kahteen suuntaan linkitetty järjestetty lista
ETSI(L,k)				
LISÄÄ(L,x)				
POISTA(L,x)				
MINIMI(L)				
MAKSIMI(L)				

Ratkaisu. Taulukkoon on listattu (suurempien määrien osalta summittaiset) operaatiomäärät, kun listassa 1000 alkiota.

	Yhteen suuntaan linkitetty järjestämätön lista	Yhteen suuntaan linkitetty järjestetty lista	Kahteen suuntaan linkitetty järjestämätön lista	Kahteen suuntaan linkitetty järjestetty lista
ETSI(L,k)	n. 2000	n. 2000	n. 2000	n. 2000
LISÄÄ(L,x)	2	n. 2000	5	n. 2000
POISTA(L,x)	n. 2000	n. 2000	4	4
MINIMI(L)	n. 2000	1	n. 2000	1
MAKSIMI(L)	n. 2000	n. 2000	n. 2000	1

Kommentteja. Oletetaan, että järjestetyt listat on järjestetty pienimmästä avaimesta suurimpaan.
Etsiminen. Tietyn avaimen etsiminen listasta vaatii aina huonoimmassa tapauksessa noin 1000 vertailua ja siirtymistä seuraavaan solmuun, olipa kyseessä minkälainen lista tahansa, koska listaan ei voi tehdä puolitushakua, vaikka se olisikin järjestetty.
Lisääminen. Jos lista ei ole järjestetty, avain voidaan lisätä listan alkuun, mikä on vakioaikainen operaatio. Yhteen suuntaan linkitetyssä listassa on päivitettävä lisättävän solmun next-osoitin entiseen päähän ja muutettava uusi pää lisättäväksi solmuksi.

Kahteen suuntaan linkitetyssä listassa pitää vielä päivittää previous-osoittimet. Jos lista on järjestetty, on avaimelle haettava paikka käymällä lista läpi. Tähän vaaditaan huonoimmassa tapauksessa noin 1000 vertailua ja siirtymistä seuraavaan solmuun.
Poistaminen. Kahteen suuntaan linkitetystä listasta voidaan alkio poistaa vakioajassa, koska alkiossa on linkki sitä edeltävään ja seuraavaan alkioon. Näin ollen nämä alkiot voidaan linkittää toisiinsa kahdella osoittimen sijoituksella (lisäksi vertailut, onko osoitin tyhjä). Sen sijaan yhteen suuntaan linkitetyssä listassa alkiosta on linkki vain seuraavaan alkioon, joten alkion edeltäjä joudutaan hakemaan listasta, mikä huonoimmassa tapauksessa vaatii noin 2000 operaatiota (jokaisessa solmussa vertailu ja mahdollinen siirtyminen). Itse poistaminen vie tässäkin tapauksessa vain muutaman operaation.
Minimi. Järjestetyistä listoista alkio löytyy listan kärjestä, joten se löytyy yhdellä operaatiolla. Järjestämättömissä listoissa joudutaan sen sijaan käymään koko lista läpi.
Maksimi. Järjestetyissä listoissa alkio on viimeisenä. Kahteen suuntaan järjestetyssä listassa on osoitin listan häntään, joten alkio löytyy yhdellä operaatiolla. Muissa tapauksissa joudutaan käymään lista kokonaan läpi. Joskus myös yhteen suuntaan linkitetyssä listassa säilytetään osoitinta listan häntään, jolloin maksimikin löytyisi järjestetystä listasta yhdellä operaatiolla.

Tehtävä 1.2 Esitä pseudokoodilla algoritmi, joka kääntää sille syötetyn yhteen suuntaan linkitetyn listan L järjestyksen päinvastaiseksi. Käytä apuna pinoa. Listan L pään osoitin on L.head ja listassa olevan solmun x kenttä x.next osoittaa solmua x seuraavaan solmuun.
Pinon operaatiot on annettu seuraavassa tehtävässä.

Ratkaisu.
Oletetaan, että käytettävissä on pino, joka on kätevä käännettäessä järjestys. Käydään läpi lista, laitetaan alkiot pinoon, otetaan pois käänteisessä järjestyksessä ja huolehditaan että listan uudeksi pääksi tulee entinen viimeinen alkio ja linkit kääntyvät. Alla on pseudokoodi. Algoritmin voi myös toteuttaa helposti ilman pinoa.

```
REVERSE(L)
1.  if L.head != NIL
2.     STACK S
3.     x = L.head
4.     while x!=NIL
5.        S.PUSH(x)
6.        x = x.next
7.     L.head = S.POP()
8.     prev = L.head
9.     while !S.EMPTY()
10.       x = S.POP()
11.       prev.next = x
12.       prev = x
13.   prev.next = NIL
```

Tehtävä 1.3 Osoita oikeiksi seuraavat väitteet:

a) pinon toiminnot voidaan toteuttaa kahdella jonolla;
b) jonon toiminnot voidaan toteuttaa kahdella pinolla.

Pinon ja jonon operaatiot ovat seuraavat:

Pino S	**Jono Q**
S.EMPTY() – True, jos S tyhjä. False muuten	Q.EMPTY() – True, jos Q tyhjä. False muuten
S.PUSH(x) – Lisää alkion x pinon S päälle	Q. ENQUEUE(x) – Lisää alkion x jonon häntään
S.POP() – Poistaa alkion pinon S päältä ja palauttaa sen	Q. DEQUEUE() – Poistaa jonon päästä ensimmäiseksi lisätyn alkion ja palauttaa sen.

Ratkaisu.

Pinon operaatiot ovat EMPTY, PUSH ja POP. Pinossa siis viimeiseksi lisätty alkio otetaan ensimmäiseksi pois.
Jonon operaatiot ovat EMPTY, ENQUEUE ja DEQUEUE. Jonossa ensimmäiseksi lisätty alkio otetaan ensimmäiseksi pois.

a) On siis toteutettava pinon operaatiot, kun käytetään kahta jonoa Q1 ja Q2. Toimitaan niin, että jono Q2 on aina tyhjä ja sitä käytetään vain apuna lisättäessä alkio pinoon. Lisättäessä laitetaan jonoa Q2 apuna käyttäen jonon Q1 häntään, jolloin se voidaan ottaa suoraan pois. Näin saadaan operaatioille pseudokoodit

```
#STACK WITH TWO QUEUES Q1 AND Q2
EMPTY()
1.  return Q1.EMPTY()

POP()
1.  return Q1.DEQUEUE()

PUSH(x)
1.  while(!Q1.EMPTY())
2.     y = Q1.DEQUEUE()
3.     Q2.ENQUEUE(y)
4.  Q1.ENQUEUE(x)
5.  while(!Q2.EMPTY())
6.     y = Q2.DEQUEUE()
7.     Q1.ENQUEUE(y)
```

Operaatiot EMPTY ja POP ovat vakioaikaisia, mutta PUSH riippuu pinossa olevien alkioiden määrästä.

b) On siis toteutettava jonon operaatiot, kun käytetään kahta pinoa S1 ja S2. Toimitaan niin, että pino S2 on aina tyhjä ja sitä käytetään vain apuna lisättäessä alkio pinoon. Lisättäessä laitetaan pinoa S2 apuna käyttäen uusi alkio pinon S1 pohjalle, jolloin se otetaan viimeiseksi pois. Pinon S1 päälle tulee jonon pää, jolloin alkio voidaan ottaa suoraan pois. Näin saadaan operaatioille pseudokoodit

```
#QUEUE WITH TWO STACKS S1 AND S2
EMPTY()
1.  return S1.EMPTY()

DEQUEUE()
1.  return S1.POP()

ENQUEUE(x)
1.  while(!S1.EMPTY())
2.     y = S1.POP()
3.     S2.PUSH(y)
4.  S1.PUSH(x)
5.  while(!S2.EMPTY())
6.     y = S2.POP()
7.     S1.PUSH(y)
```

Operaatiot EMPTY ja DEQUEUE ovat vakioaikaisia, mutta ENQUEUE riippuu jonossa olevien alkioiden määrästä.

Tehtävä 1.4. Erään palvelupisteen jonoon voidaan ottaa korkeintaan 10 asiakasta. Kirjoita (C- tai Python-kielellä) ohjelma, jossa voidaan lisätä asiakkaita yksi kerrallaan jonoon ja ottaa asiakkaita jonosta yksi kerrallaan saapumisjärjestyksessä. Asiakkaat numeroidaan juoksevalla järjestysnumerolla. Jos jonossa on jo 10 asiakasta, uutta asiakasta ei voida lisätä. Käytä ohjelmassasi hyväksi jono-tietorakennetta. Lataa ohjelman pohjaksi alle linkitetty tiedosto task4_base.c tai task4_base.py.

Ratkaisu. Ratkaisuohjelmat on linkitetty alle. Kummassakin jono toimii taulukolla toteutettuna rengaspuskurina, kuten luennoissa on esitetty. C-ohjelmassa tietorakenne on toteutettu structina queue, jota käyttävät funktiot enqueue() ja dequeue(). Edellinen palauttaa arvon 0, jos jonoon ei voi lisätä enää alkioita; jälkimmäinen palauttaa alivuodon tapauksessa arvon -1, joka ei voi olla asiakkaan numero. Pääohjelmassa otetaan paluuarvot huomioon. Python-ohjelmassa tietorakenne on toteutettu luokkana, jossa on attribuuttina myös taulukon koko. Operaatiot enqueue() ja dequeue() on toteutettu C-ohjelman tapaan funktioina eikä luokan metodeina, mikä olisi myös ollut mahdollista (ja myös olio-ohjelmointiparadigman mukaista). Huomaa vielä, että enqueue() palauttaa arvon True/False sen mukaan, onnistuiko alkion lisääminen. Muuten toiminta on samanlainen kuin C-ohjelmassa.

Kummallakin kielellä ohjelman olisi luonnollisesti voinut toteuttaa myös niin, että jonona olisi käytetty pääohjelmassa paikallisena muuttujana olevaa taulukkoa.