RĪGAS TEHNISKĀ UNIVERSITĀTE

Datorzinātnes un informācijas tehnoloģijas fakultāte

Lietisko datorsistēmu institūts

REFERĀTS

Mājas darbs mācību priekšmetā
„**Datu struktūras**”

Vienkāršsaistītu un divkāršsaistītu sarakstu salīdzinājums.

Izstrādāja: Vitālijs Hodiko

RDBF0 8. grupa

Pārbaudīja: asoc. Profesore N.Prokofjeva

Inv. Nr.	Datums	Atzīme

2010./2011. mg.

ANOTĀCIJA

Šaja referata ir apskatits divkāršsaistīta un vienkāršsaistīta saraksta salidzīnajums. Ir uzrakstītas programmas kas 32727 elementiem katram saraksta tipam izpilda sekojošas operacijas: jauna elemena pievienošana saraksta sākumā, vidū un beigās, patvaļiga elementa nolāsīšana, elementa iznicinašana.

SATURS

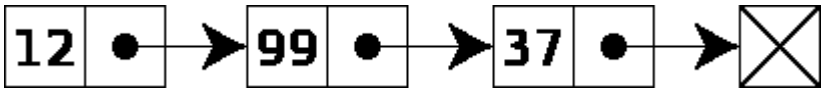
1 IEVADS

Lai glabātu un ātri apstrādāt lielo datu apjomu ir vērts lietot dinamiska datu struktūras. Pirmais no tiem ir vienkāršsaistīts - datu struktūra, kas sastāv no secīgām mezglu no kuriem katrs satur atsauci (t.i. saiti) uz nākamo mezglu secībā (eng - data structure that consists of a sequence of nodes each of which contains a reference (i.e., a link) to the next node in the sequence) un divkāršsaistīts - datu struktūra, kas sastāv no secīgām mezglu no kuriem katrs satur atsauci (ti, saiti) uz nākamo un iepriekšējo mezglu secībā (eng - data structure that consists of a sequence of nodes each of which contains a reference (i.e. a link) to the next and previous node in the sequence) saraksts. Tādas datu struktūras ir ierobežotas tikai ar pieejamo operatīvas atminas apjomu. Tā ka tādas datu struktūras glabājas operatīva atmina nevis uz cieta diska vai kāda cita datu nesēja tad to ātrdarbība ir loti liela.

2 VIENKĀRŠSAISTĪTĀ UN DIVKĀRŠSAISTĪTĀ SARAKSTA SALIDZINĀŠANA

2.1 Sarakstu apraksti

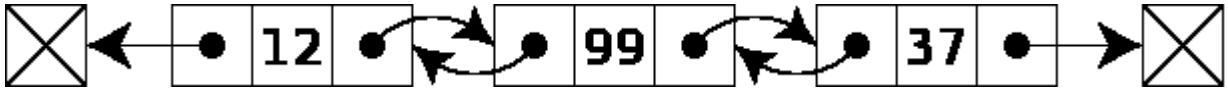
2.1.1 Vienkāršsaistītāis saraksts



2.1.1.1 att: Vienkāršsaistītā saraksta attēlošana

Katrs elements glaba atsauci uz nākamo elementu. Tas nozīmē, ka ir dažas pārterejumi (uzglabāt atsauci uz nākamo elementu). Arī tāpēc, ka viņi nav saglabāti pēc kārtas, nevar uzreiz doties uz 657415671567 elementu - jums ir jāsāk no galvēna elementa (1 elements, head), un pēc tam saņemt atsauci, lai dotos uz 2, un pēc tam saņemt nākamo, lai nokļūt uz trešo, ... un pēc tam saņemt tā atsauci, lai nokļūtu uz 657415671566, un tikai pēc tam tiksiet lidz 657415671567 elementa. Šādā veidā tas ir ļoti neefektīvs, lai pārskatītu to elementus ne pēc kārtas, jeb gadijuma elementu. Tomēr tas ļauj mainīt saraksta garumu. Ja jūsu uzdevums ir parskatit katru elementu pēc kārtas, tad tas ir apmēram tāds pats kā vienkāršs saraksts. Ja jums ir nepieciešams mainīt saraksta garumu, tas jau varētu būt labāks nekā vienkāršs sarakstā. Ja jūs zināt 566. elements, un jūs meklējat 567. elementu, tad viss, kas jums jādara, ir sekot atsaucei uz nākamo. Tomēr, ja jūs zināt 567. un jūs meklējat 566. elementu, vienīgais veids, kā atrast to, ir sākt meklēšanu no 1 elementa. Un šajā vieta divkāršsaistītāis saraksts ir ērtāks.

2.1.2 Divkāršsaistītāis saraksts



2.1.2.1 att: Divkāršsaistītā saraksta attēlošana

Divkāršsaistītāis saraksts glāba atsauci uz iepriekšējo elementu. Tas nozīmē, ka jūs varat parvietoties sarakstā kā atpakaļ, tā arī uz priekšu. Tas varētu būt ļoti noderīgi dažos gadījumos. Izņemot šo, divkāršsaistītā saraksta priekšrocības un trūkumi ir gandrīz tadi paši kā vienkāršsaistītā saraksta.

2.2 Sarakstu teorētiskai salīdzinājums

2.2.1 Vienkāršsaistītāis lineārais saraksts vs citi saraksti

Lai gan divkāršsaistītāis saraksts un cirkularais saraksts ir labāks variants nekā vienkāršsaistītāis lineārais saraksts, lineārs saraksts piedāvā dažas priekšrocības, kas padara tos vēlamus dažās situācijās.

No vienas puses, vienkāršsaistītāis lineārais saraksts ir rekursīva datu struktūra, jo tā satur rādītāju uz mazāku elementu ta paša tipa. Šī iemesla dēļ vairākas darbības ar šo sarakstu (piemēram, apvienot divus sarakstus, vai uzskaitit elementus apgrieztā secībā), bieži vien prasa ļoti vienkāršu rekursīvu algoritmu, daudz vienkāršāku nekā, izmantojot iteratīvas komandas. Kaut arī var pielāgot šīs rekursīvas risinājumus divkāršsaistītām un cirkulāri saistītām sarakstiem, procedūras parasti prasa papildus argumentus un daudz sarežģītākas pamata gadījumos.

Vienkāršsaistītāis lineārais saraksts ļauj arī, tā saucamo 'tail-sharing', izmantojot apakšsaraksta kopēju galīgo daļu, kā gala daļu diviem dažādiem sarakstiem. Jo īpaši, ja jauns mezgls ir pievienots saraksta sākumā, iepriekšejais saraksts ir pieejams kā 'tail' jaunājā sarakstā - vienkāršs piemērs datu struktūras. Atkal, tā nav taisnība, ar citiem variantiem: mezgls var arī nepiederēt divām dažādām cirkulārām vai divkāršsaistītiem sarakstiem.

Jo īpaši gala kontrolpunktu mezgli var dalīti starp vienkāršsaistītāis necirkulariem sarakstiem. Var pat izmantot to pašus gala kontrolpunktu mezglus katram šādam sarakstam. Lisp, piemēram, katrs raksturīgs saraksts beidzas ar saiti uz īpašu mezglu, apzīmētu ar null, kura 'CAR' un 'CDR' saista ar sevi. Tādējādi Lisp procedūra var droši ņemt 'CAR' vai 'CDR' no jebkura saraksta.

Patiešām, moderno variantu priekšrocības bieži vien aprobežojas ar algoritma sarežģītību, bet ne to efektivitāti. Cirkularais saraksts, parasti var būt atdarināts ar lineāru sarakstu kopā ar diviem mainīgajiem, kas norāda uz pirmo un pēdējo mezglu, bez papildu izmaksām.

2.2.2 Divkāršsaistītāis vs vienkāršsaistītāis saraksts

Divkāršsaistītām sarakstam ir nepieciešams vairāk vietas uz vienu mezglu (izņemot, ja izmanto XOR-linkošanu), un to pamata darbībās ir dārgākas, bet tās bieži vien ir vieglāk manipulēt, jo tie ļauj secīgi piekļūt sarakstam abos virzienos. Tas nozīme, ka var ievietot vai dzēst mezglu ar pastavigu operāciju skaitu, izmantojot tikai ta mezgla adresi. Lai veiktu to pašu, vienkāršsaistītā sarakstā, vajag zināt iepriekšējā elementa adresi. Dažiem algoritmiem ir vajadzīga piekļuve abos virzienos. No otras puses, divkāršsaistīti saraksti neļauj 'tail' sadali un to nevar pastāvīgi izmantot datu struktūras.

2.3 Praktiskais salidzinājums

2.3.1. Tabula: Salīdzinājuma dati

Darbības	Elementu skaits	Laiks	
		DLL	SLL
Jauna el. pievienošana	32767		
Vidus		14.561	19.014
Beigās		0.016	0.016
Sakumā		0.016	0.016
Gadijuma pozicija		5.953	28.918
Nolasīšana			
Gadijuma elementa		14.593	32.968
Elementa izdzēšana			
No beigas		0.016	0.16
No vidus		15.468	33.875
No sakuma		0.016	0.016
No gadijuma pozicijas		4.312	36.936

Kā redzams dažas operācijas divkāršsaistītā sarakstā un vienkāršsaistītā sarakstā izpildās vienādi ilgi, ja darbības notiek bez pārvietošanas saraksta, bet ja nepieciešams pārvietoties sarakstā, tad divkāršsaistīts saraksts ir ievērojami ātrāks. Ir gadījumi, kad divkāršsaistīts saraksts ir gandrīz 10 reizes ātrāks. Protams par ātrumu ir jāmaksā, šajā gadījumā katrs divkāršsaistīta saraksta elements aizņem par 4 baitiem vairāk nekā vienkāršsaistīta saraksta elements.

SECINĀJUMI

Analizējot iepriekš minēto var izdarīt secinājumus kā neskatoties uz to, kā divkāršsaistītām sarakstam nepieciešams izdarīt vairāk operāciju jauna elementa pievienošanai, tas ir ievērojami ātrāks nekā vienkāršsaistīts saraksts. Mūsdienīgos datoros operatīvas atmiņas apjoms ir pietiekami liels, lai mēs varētu neņemt vērā, ka vienkāršsaistīta saraksta viens elements aizņem atmiņā par 4 baitiem mazāk nekā divkāršsaistīta saraksta elements.

BIBLIOGĀFIJA

- http://en.wikipedia.org/wiki/Linked_list#Singly_linked_linear_lists_vs._other_lists
- http://stackoverflow.com/questions/712429/plain-linked-and-double-linked-lists-when-and-why