ივანე ჯავახიშვილის სახელობის თბილისის სახელმწიფო უნივერსიტეტი
ზუსტ და საბუნებისმეტყველო მეცნიერებათა ფაკულტეტი

კომპიუტერული მეცნიერების დეპარტამენტი

ნიკოლოზ გრძელიძე

მესამე საკონფერენციო მოხსენება

მონაცემთა ჰიბრიდული სტრუქტურები

თბილისი 2015

სარჩევი

1 მონაცემთა ჰიბრიდული სტრუქტურები

1.1 ნაშრომის დასახელება. "ჰიბრიდული მონაცემთა სტრუქტურები" წარმოიშვა ტერმინიდან "მრავალრეჟიმიანი ხე", რომელიც შემოხებული იყო ჩემს სამაგისტრო ნაშრომში, 2013 წელს. აღნიშნული ტერმინი წარმოადგენს მისი მშობელი ტერმინის განზოგადებას. ჰიბრიდული სტრუქტურა იძენს მისი შემადგენელი მარტივი მონაცემთა სტრუქტურების საუკეთესო თვისებებს და დამატებით კიდევ მარტივი სტრუქტურების ერთმანეთში სწრაფი ტრანსფრომაციის, საშუალებას

ჩემ შემთხვევაში, ჰიბრიდული სტრუქტურა წარმოადგენს ორი მარტივი სტრუქტურის RBT და AVL- ის გაერთიანებას. ნაშრომში წარმოდგენილი ჰიბრიდულ სტრუქტურას აქვს ორი მდგომარეობა, ის ან RBT ან AVL_ია , შესაბამისად ორი სახის ტრანსფორმაცია გვაქვს RBT_დან AVL_ში და AVL_დან RBT_ში.

1.2. სამეცნიერო კვლევის აქტუალობა. თავდაპირველად აღვნიშნავ, რომ მრავალრეჟიმიანი ძებნის ორობითი ხის გამოყენება მცირე ზომის მონაცემებზე

არასაჭიროდ მიმაჩნია, რადგან განსხვავების დანახვა ამ შემთხვევაში რთულია, თუმცა დიდი მონაცემების შემთხვევაში ნამდვილად სასარგებლო იქნება მისი გამოყენება და ბევრად აჩქარებდა მონაცემთა ძებნის პროცესი, მაშასადამე დგას საჭიროება მრავალრეჟიმიანი ძებნის ორობითი ხის გამოყენებისა.
საჭიროებიდან გადავიდეთ თემის აქტუალობაზე, აქტუალობას განვსაზღვრავ RBT და AVL ხეების აქტუალობიდან გამომდინარე და გამოვრიცხავ შემთხვევას, როცა ჩემი ნაშრომის აქტუალობა მათზე ნაკლები იქნება.

სწრაფი მონაცემთა სტრუქტურების არსებობა მნიშვნელოვანი ნაწილია მაღალი დონის პროგრამული უზრუნველყოფისთვის, შესაბამისად შეგვიძლია მათი რეალიზაცია გადმოვიტანოთ ჰიბრიდულ მონაცემთა სტრუქტურაზე. გალითისთვის შეიძლება მოვიყვანოთ RED-BLACK TREE, რომლის რეალიზაცია Java_ზე არის მონაცემთა სტრუქტურა TreeMap, რომელიც ერთერთ ხშირად გამოყენებად კონტეინერს წარმოადგენს.

1.3. არსებული გამოცდილება

სამაგისტრო ნაშრომის ფარგლებში, შემოღებული იქნა კვანძის საერთო სტრუქტურა რომელიც საერთოა ორივე მონაცემთა სტრუქტურისათვის. გამოყენებულია კვანძის ჩასმის ექსპერიმენტალური მეთოდი, რომელიც ამცირებს მუშაობის დროს მიმდევრობით შემოსული მონაცემისათვის.

1.4. კვანძის სტრუქტურა: გამომდინარე თემის სახელიდან, აუცილებელია გამოვიყენოთ კვანძის სტრუქტურა რომელიც აღწერს როგორც AVL ისე RB ხის კვანძს. სტრუქტურა შედგება ხუთი ველისაგან ესენია: key, parent, left, right, staff. თითოეულ ველს სრულად იყენებს ორივე სტრუქტურა, პირველი ოთხი ველის დანიშნულება ორივე სტრუქტურაში იდენტურია, რაც შეეხება staff ველს RB_ს შემთხვევაში მასში ფერის მნიშვნელობა ინახება, AVL ის შემთხვევაში ხის სიმაღლე.

1.5. ორობითი ხის სტრუქტურა. რადგან კვანძის ჩასმის ორი რეჟიმი გვაქვს (განვიხილავ მოგვიანებით) საჭიროა დამატებითი სტრუქტურის შექმნა რომელიც შეიცავს ძებნის ორობითი ხის ფესვს და მაქსიმალურ ელემენტს. რეალიზაციაში სტრუქტურის სახელია BSTTree, რომელიც შეიცავს კვანძის ტიპის ორ ველს root და max. მრავალრეჟიმიანი ძებნის ორობითი ხის აგებისათვის max ველი გამოყენებული იქნება ხის შემოვლის დაწყების წერტილად.
ძებნის ორობით ხეებზე დაფუძნებულ ყველაზე ხშირად გამოყენებად მონაცემთა სტრუქტურებს წარმოადგენენ, წითელ-შავი (შემდეგში RB), Adelson-Velskii and Landis' tree (შემდეგში AVL) და Splay ხეები. აღნიშნული მონაცემთა სტრუქტურები ხშირად გამოიყენება სისტემურ პროგრამულ უზრუნველყოფაში, როგორიცაა ოპერაციული

სისტემის ბირთვი. რათქმაუნდა, არსებობს სხვა კონკრეტული რეალიზაციები, მაგრამ მათ არ განვიხილავთ ნაშრომის ფარგლებში. სამი ძირითადი ოპერაცია -კვაძის ჩამატება,წაშლა და ძებნა, რომლებიც წარმოადგენენ BST შედარების კრიტერიუმებს, განსაზღვრავს მისი სტრუქტურის პერფომანსს. აღნიშნულ კრიტერიუმებზე დაყრდნობით RB და AVL ხეებს გააჩნიათ უპირატესობა სტანდარტულ BST-სთან შედარებით, ცრილში მოცემულია სტანდარტული BST, RBT და AVL-ის შედარებები უარეს შემთხვევებში.

	ჩასმა	წაშლა	ძებნა
BST	O(n)	O(n)	O(n)
RBT	O(log n)	O(log n)	O(log n)
AVL	O(log n)	O(log n)	O(log n)

რაც შეეხება საშუალო შეფასებას, აღნიშნული ოპერაციებისთვის, სამივე ტიპის სტრუქტურისათვის ერთიდაიგივეა -O(log n).

1.6. კორექტულობა: არჩეული გზის კორექტულობის დასამტკიცებლად, ნაშრომის პროგრამულმა რეალიზაციას დაემატე, სტანდარტული იმპლემენტაციები (კორმენის მიხედვით) RB და AVL ხეებისაათვის. ასევე დაემატებულია ტესტირების ავტომატური მეთოდები. რეალური ტესტირება გადამწყვეტია არჩეული გზის კორექტულობის გასამყარებლად.

2. დაბალანსების სხვა მეთოდები.

2.1. DSW ალგორითმი

1976 წელს Colin Day-მ შეიუმუშავა ორობითი ხის დინამიკური დაბალანსების ალგორითმი, რომელიც თავიდან ირიდებს ოპერაციების გარკვეულ რაოდენობას, რასაც

AVL გვთავაზობს. (თუმცა ის მაინც რჩება ორობითი ხის ბალანსირების დინამიკურ ალგორითმად). Day-ის ალგორითმის პირველი რეალიზაცია შესრულებული იყო fortran-ზე. რეკურსიის და კლასების უქონლობამ თავისებურად გაართულა იგი. ალგორითმი შედგება ორი ფაზისგან:

1. ორობითი ხისაგან გადაგვარებული ხის აგება - ნებისმიერ კვანძს ჰყავს მხოლოდ და მხოლოდ მარჯვენა შვილი;

2. გადაგვარებული ხისაგან დაბალანსებული ხის აგება - compression მეთოდის გამოყენებით.

გადაგვარებული ხის მისაღებად Day იყენებდა ბმულ სიას, რომელიც მოითხოვდა დამატებით მეხსიერებას. პროგრამირების ენების განვითარებასთან ერთად გაჩნდა ახალი შესაძლებლობები და ალგორითმმაც ცვლილება განიცადა.

10 წლის შემდეგ Quentin F. Stout-მა და Bette L. Warren-მა ცვლილებები შიტანეს Colin Day-ის ალგორითმში. ძირეული ცვლილება განიცადა ალგორითმის პირველმა ფაზამ და მცირედი - მეორემ. Stout-მა და Warren-მა შენიშნეს, რომ პირველ ფაზაში, გადაგვარებული ხის აგება შესაძლებელია ტრიალების გამოყენებით, კერძოდ, მარჯვნივ მობრუნებების ოპერაციით. მათ იმპლემენტაციაში შემოიღეს ფსევდო-ფესვი, რაც ალგორითმს ხდის უფრო მარტივს და თავიდან გვარიდებს განშტოების ზედმეტ ოპერაციებს. მარჯვნივ ტრიალი ხორციელდება მანამ, სანამ კვანძის მარცხენა შვილი არ გახდება null. ამის შემდეგ ალოგირთმი ჩადის ერთი დონით დაბლა და იმეორებს იგივე ოპერაციას. მიღებულ სტრუქტურას ავტორებმა უწოდეს vine tree აღნიშნულ მეთოდს კი - tree_to_vine.

ალოგირითმის მეორე ფაზაში ხდება მობრუნებების (მარცხნივ მობრუნება) ოპტიმალური K რაოდენობების დათვლა. K არგუმენტად გადაეცემა ფუნქციას, რომელიც K ჯერ მოახდენს vine tree-ს მობრუნებას მარცხნივ, რის შედეგადაც მიიღება დაბალანსებული ორობითი ხე.

ალგორითმის პირველი ფაზა, tree_to_vine ფსევდო-კოდი:

1. tmp = root // დროებითი ცვლადი. თავიდან დგას ფესვზე
2. while (tmp != NULL)

- თუ tmp ჰყავს მარცხენა შვილი

მარჯვნივ მობრუნება ამ წვეროსთვის

tmp-ს მიანიჭე მარცხენა შვილი, რომელიც ახლა მშობელი გახდა

- else: tmp-ს მიანიჭე მარჯვენა შვილი

როცა ხეში n ცალი წვეროა: საუკეთესო შემთხვევაში, როცა ხე უკვე vine-ია, ციკლი სრულდება n-ჯერ და არცერთი მობრუნება არ სრულდება. უარეს შემთხვევაში კი, როცა ფესვს მარჯვენა შვილი არ ჰყავს - ციკლი სრულდება 2n - 1 ჯერ და აკეთებს n - 1 მობრუნებას. პირველი ფაზის დროითი შეფასება გამოდის O(n).

ალგორითმის მეორე ფაზა, vine_to_tree ფსევდოკოდი:

$m = 2^{\lfloor \lg(n+1) \rfloor} - 1;$

გააკეთე m - n ცალი მობრუნება vine-ის თავიდან

while (m > 1)

m = m / 2;

გააკეთე m ცალი მობრუნება vine-ის თავიდან

მეორე ფაზის სირთულე რომ გამოვთვალოთ, დავაკვირდეთ, რომ ციკლის მიერ შესრულებული მობრუნებები ტოლია:

$(2^{\lg(m+1)-1} - 1) + \ldots + 15 + 7 + 3 + 1 = \sum_{i=1}^{\lg(m+1)-1} (2^i - 1) = m - \lg(m+1)$

მობრუნებების რაოდენობა შეგვიძლია წარმოვადგინოთ ფორმულით:

$n - m + (m - \lg(m + 1)) = n - \lg(m+1) = n - \lfloor \lg(n+1) \rfloor$

DSW ალგორითმის დროითი შეფასება გამოდის O(n).

3. კვლევის მიზანი

სადოქტორო დისერტაციის ფარგლებში იგეგმება სხვადასხვა ჰიბრიდული მონაცემთა სტრუქტურის განსაზღვრა, იმპლემენტირება და გამოყენება პრაქტიკულ ამოცანებში. ერთ-ერთ მათგანში ვფიქრობთ დაბალანსებული ძებნის ორობითი ხის და ჰეშ-ცხრილის ფუნქციების გაერთიანებას.

ასეთი სტრუქტურა ძალიან სასარგებლო იქნება მეჩხერი (sparse) მარტიცების გადამრავლებითვის და ასეთი მატრიცის ხარისხების განსაზღვრისთვის.

თვითონ მეჩხერი მატრიცების გადამრავლება საკმაოდ იშვიათად გამოიყენება პრაქტიკაში, ბევრად უფრო ხშირად გამოყენებადია მეჩხერი მატრიცის (ჩვეულებრივ) ვექტორზე გადამრავლების ოპერაციები, რის გამოც შესაბამისი ალგორითმები კარგადაა დამუშავებული.

თუმცა, ჩვენ შეგვიძლია მივუთითოთ რამდენიმე პრაქტიკულად საინტერესო ამოცანა, სადაც ასეთი აუცილებლობა ჩნდება.

ცნობილია, რომ მატრიცების ახარისხების საჭიროება ჩნდება, როდესაც გვინდა ორიენტირებული გრაფის ყველა წვეროებს შორის უმოკლესი მანძილების განსაზღვრა. მატრიცების ახარისებაზე დაფუძნებული ალგორითმი არაა ყველაზე სწრაფი, თუმცა საკმაოდ ცნობილია და მისი აჩქარება შესაძლებელია ახალი სტრუქტურის გამოყენეით ისეთი გრაფებისთვის, რომლებშიც წიბოების რაოდენობა შორსაა შესაძლო მაქსიმუმისგან (მაგალითად, პლანარული გრაფები).

მეორე, ბევრად უფრო ბუნებრივი გამოყენება შეგვიძლია მოვიყვანოთ ბიოლოგიიდან. ესაა ლესლის მატრიცა, ძალიან მპოპულარული მოდელი პოპულაციათა ეკოლოგიაში. ამ მატრიცის ხარისხები აღწერენ პოპულაციის მნიშვნელოვანი მახასიათებლების ცვლილებას დროში. მატრიცა უმეტესწუილად ნულებისგან შედგება, ამიტომ ჩვენი მიდგომა აქ პერსპექტიოული ჩანს.

მოკლედ აღვწეროთ თუ რა განასხვავებს ჩვენს მიდგომას უკვე არსებული მეთოდებისგან, რომლებიც გამოიყენება მეჩხერი, ანუ გაიშვიალლებული მონაცემების დამუშავებისთვის.

ერთ-ერთი ყველაზე სანდო ბიბლიოთეკა არის boost -ბიბლიოთეკა. იგი ამ მიზნიტ იყენებს წითელ-შავი ხის საფუძველზე შექმნილ მეჩხერი მატრიცის ადაპტერს, რომელიც მარტიცის ვექტროზე გამრავლებისთვის, მატრიცის სტრიქონებს გაივლის წრფიც დროში იტერატორის გამოყენებით, ხოლო სტრიქონის ელემენტების ამოღება დამოუკიდებლად ხდება. როდესაც მარტიცებს ვამრავლებთ ერთმანეთზე, მატრიცის სტრიქონები წრფივ დროში გაისინჯება (და ნულოვანი ელემენტები გამოიტოვება), მაგრამ ამავე დროს, სვეტების ელემენტების ამოსაღებად საჭიროა შენახული ელემენტების როადენობის ლოგარითმის პროპორციული დრო. შევნიშნოთ, რომ აქ ეკონომიას ვერ ვაღწევთ, ყველა ელემენტი უნდა მოვძებნოთ, არის ჩაწერილი თუ არა. იმ შემთხვევაში, როდესაც ვიყენებთ ჰიბრიდს, ელემენტის ძებნა ხორციელდება მუდმივ დროში, რაც გვაძლევს სასურველ საგრძნობ განსხვავებს პროცედურის მრავლალჯერ განხორციელების შემთხვევაში. ჰიბრიდის სტრუქტურა საკმაოდ რთულია და ამიტომ მასზე აქ არ შევჩერდებით.